CS 194: Distributed Systems
Distributed based Object Systems

Scott Shenker and Ion Stoica
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1776

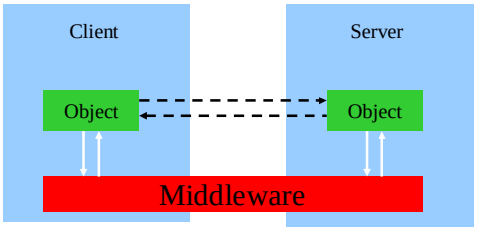
Outline

- Common Object Request Broker Architecture (CORBA)
- Distributed Common Object Model (DCOM)

Introduction to CORBA

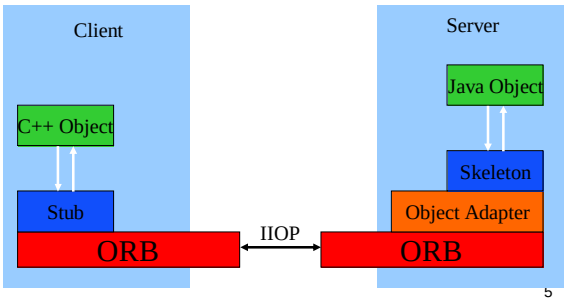
- The Object Management Group (OMG) was formed in 1989. Its aims were:
  - to make better use of distributed systems
  - to use object-oriented programming
  - to allow objects in different programming languages to communicate with one another
- The object request broker (ORB) enables clients to invoke methods in a remote object
- CORBA is a specification of an architecture supporting this.
  - CORBA 1 in 1990 and CORBA 2 in 1996.

Generic Architecture



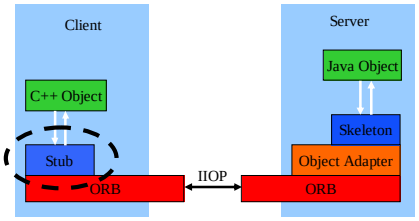
CORBA Architecture

- Remote-object: object implementation resides in server's address space



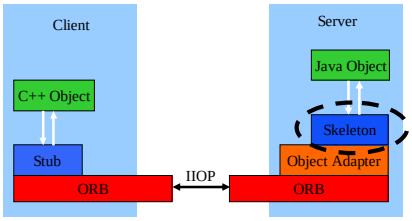
Stub

- Provides interface between client object and ORB
- Marshalling: client invocation
- Unmarshalling: server response



Skeleton

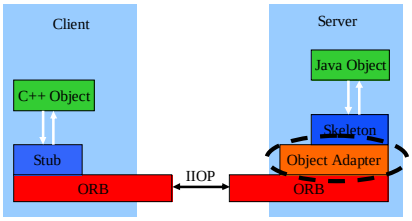
- Provides iterface between server object and ORB
- Unmarshaling: client invocation
- Marshaling: server response



7

(Portable) Object Adapter (POA)

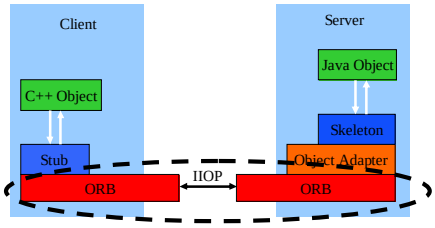
- Register class implementations
- Creates and destroys objects
- Handles method invokation
- Handles client authentication and access control



8

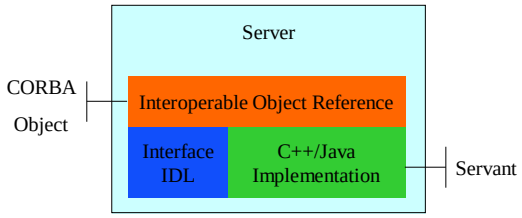
Object Request Broker (ORB)

- Communication infrastructure sending messages between objects
- Communication type:
  - GIOP (General Inter-ORB Protocol)
  - IIOP (Internet Inter-ORB Protocol) (GIOP on TCP/IP)



9

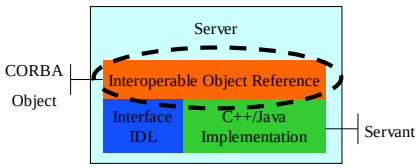
CORBA Object



10

Interoperable Object Reference (IOR)

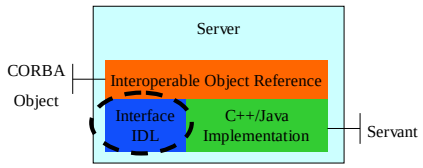
- **Uniquely identifies an object**
- **Example**
  - **IOR:000000000000001049444c3a5472697669616c3a312e300000000001**
    0000000000000007c000102000000000d3135322e38312e342e3131300000
    048000000025abacab313130303338363231333600 5f526f6f74504f41000
    0cafebabe3bd5b87800000000000000000000010000001000002c00000
    0000010001000000040001002000010109000101000501000100010109 0
    0000002000101000501000 1



11

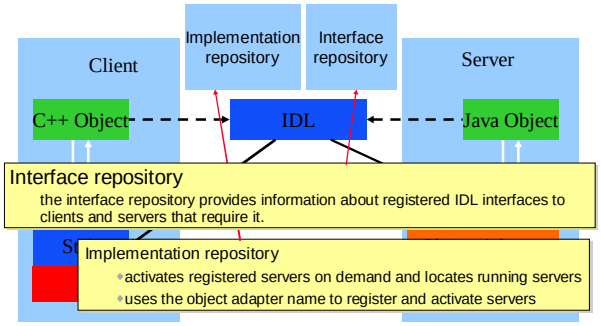
Interface Definition Language (IDL)

- Describes interface
- Language independent
- Client and server platform independent



12

Overall CORBA Architecture



13

Example of CORBA Services

- Naming: Keeps track of association between object names and their reference. Allows ORB to locate referenced objects
- Life Cycle: Handles the creation, copying, moving, and deletion objects
- Trader: A "yellow pages" for objects. Lets you find them by the services they provide
- Event: Facilitates asynchronous communications through events
- Concurrency: Manages locks so objects can share resources
- Query: Locates objects by specified search criteria
- …

14

Object Invocation Models

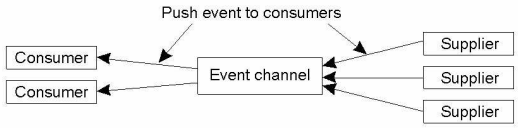
- Invocation models supported in CORBA

Request type	Failure semantics	Description
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

15

Event and Notification Services (1)

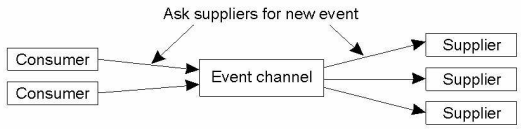
- Push model
- Each event is associated with a single data item
- Events are delivered through a channel
- Consumers need to register to a channel



16

Event and Notification Services (2)

- Pull model

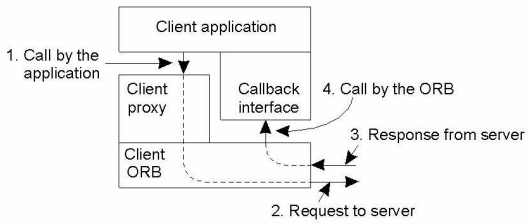


17

Messaging (1)

- Asynchronous method invocation
- Example:

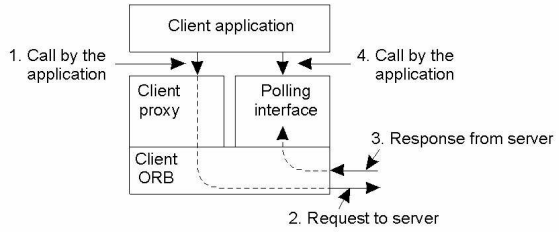
```
void sendcb_add(in int i, in int j);    // called by client
void replycb_add(in int ret_val, in int k); // called by client's ORB
```



Messaging (2)

- Pulling model for asynchronous method invocation
- Example:

```
void sendpoll_add(in int i, in int j);    // called by client
void replycall_add(out int ret_val, out int k); // called by the client
```



Interoperability

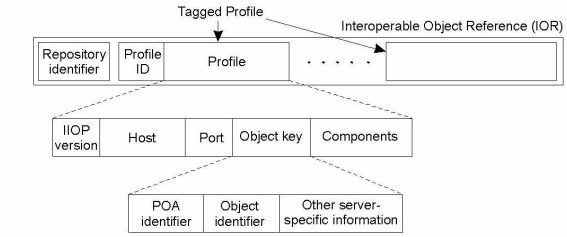
- Allow multi-vendor ORB implementations to communicate with each other
- General Inter-ORB Protocol (GIOP) message types

Message type	Originator	Description
Request	Client	Contains an invocation request
Reply	Server	Contains the response to an invocation
LocateRequest	Client	Contains a request on the exact location of an object
LocateReply	Server	Contains location information on an object
CancelRequest	Client	Indicates client no longer expects a reply
CloseConnection	Both	Indication that connection will be closed
MessageError	Both	Contains information on an error
Fragment	Both	Part (fragment) of a larger message

20

Object References (1)

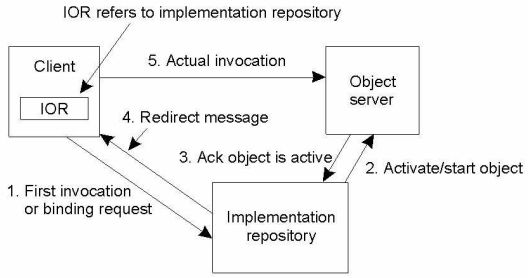
- The organization of an IOR with specific information for IIOP



21

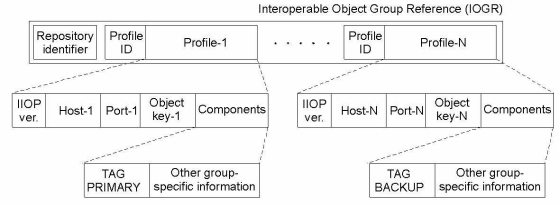
Object References (2)

- Indirect binding in CORBA



Fault Tolerance: Object Groups

- Object groups: one or mode identical copies of same object
- Replication transparent to client
- Replication strategies
  - Primary-backup, Quorum, ...



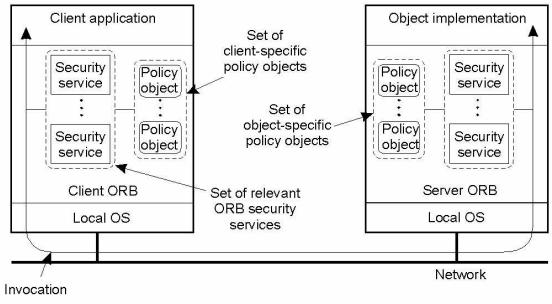
23

Security

- Transparency: application-level objects should be unaware of security services which are used
- Control: client/object should be able to specify security requirements
- Security polices: specified by policy objects
- Administrative domain where client/server is executed determines set of security services

24

Secure Object Invocation in CORBA



25

CORBA Application

1) Define interface using IDL
2) Compile interface
3) Implement interface
4) Instantiate server:
   - Register object as a CORBA object
5) Instantiate client:
   - Invoke CORBA object

- Example using a Java client and server

26

CORBA IDL interfaces Shape and ShapeList

```
struct
Rectangle{
    long width;
    long height;
    long x;
    long y;
} ;
interface Shape {
    long getVersion() ;
    GraphicalObject getAllState() ;
};

typedef sequence <Shape, 100> All;
interface ShapeList {
    exception FullException{ };
    Shape newShape(in GraphicalObject g) raises (FullException);
    All allShapes();
                    // returns sequence of remote object references

struct GraphicalObject
{
    string type;
    Rectangle enclosing;
    boolean isFilled;
                    // returns state of the GraphicalObject
```

this *struct* is used in defining another *struct*.

this *struct* is used as a parameter or result type in methods in the remote interfaces

an interface specifies a name and a set of methods

sequences and arrays in *typedefs*

interface *ShapeList*

the parameter of *newShape* is an *in* parameter and of type *Graphical Object* The return value is an extra *out* parameter of type *Shape*. No classes can be passed as arguments or results

Exceptions defined by *raises* and set by *throw*. They can have arguments.

IDL Interface

- The interface compiler is called *idltojava*
- When given an IDL interface, it produces
  - Server skeletons for each class (e.g. *_ShapeListImplBase*)
  - Proxy classes (e.g. *_ShapeListStub*)
  - A Java class for each *struct* e.g. *Rectangle*, *GraphicalObject*
  - Helper classes (*narrow* method) and holder classes (for *out* arguments)
  - The equivalent Java interfaces (e.g. *ShapeList* below)

28

The *ShapeListServant* class of the Java server program for the CORBA interface *ShapeList*

```
import org.omg.CORBA.*;
class ShapeListServant extends _ShapeListImplBase {
    ORB theOrb;
    private Shape theList[];
    private int version;
    private static int n=0;
    public ShapeListServant(ORB orb){
        theOrb = orb;
        // initialize the other instance vari
    }
    public Shape newShape(GraphicalObject g) throws ShapeListPackage.FullException {
        version++;
        Shape s = new ShapeServant( g, version);
        if(n >=100) throw new ShapeListPackage.FullException();
            theList[n++] = s;
            theOrb.connect(s);
            return s;
    }
    public  Shape[] allShapes(){ ... }
    public int getVersion() { ... }
}
```

- A Java server has classes for its IDL interfaces (e.g. *Shape* and *ShapeList*). Here is the class *ShapeListServant*

A servant class extends the corresponding skeleton class (e.g. *ShapeListImplBase*)

CORBA objects are instances of servant classes.

A servant class implements the methods in the interface (*ShapeList*). *newShape* is a factory method. It creates new CORBA objects. It uses the *connect* method to inform the ORB about the new CORBA object. (it has a remote reference module)

Java class *ShapeListServer* (the server class)

```
import org.omg.CosNaming.*;
                                    class contains the main method
                                    .*;
                                    it creates and initialises the ORB
                                    eListServant(orb);
        org.omg.CORBA.Object objRef =
            orb.resolve_in
        NamingContext nc
        NameComponent
        NameComponent
        ncRef.rebind(path, shapeRef);
        java.lang.Object sync = new java.lang.Object();
        synchronized (sync) {   sync.wait();}
    } catch (Exception e) { ... }
  }
}
```

1. it gets a reference to the Naming Service
2. narrows it to *NamingContext*- from *Object*
3. makes a *NameComponent* containing the name "*ShapeList*"
4. makes a path
5. uses *rebind* to register the name and object reference

it creates an instance of *ShapeListServant* class - a Java object - which is made a CORBA object by using the *connect* method to register it with the ORB

it waits for client requests

30

Java client program for CORBA interfaces *Shape* and *ShapeList*

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.Nam
import org.omg.CORBA.*;
public class ShapeListClient{
    public static void main(String
        try{
            ORB orb = ORB.init(args, null);
            org.omg.CORBA.Object objRef =
            ShapeList shapeListRef =
                ShapeListHelper.narrow(ncRef.resolve(path));
            Shape[] sList = shapeListRef.allShapes();
            GraphicalObject g = sList[0].getAllState();
        } catch(org.omg.CORBA.SystemException e) {...}
    }
```

1. it contacts the *NamingService* for initial context
2. Narrows it to *NamingContext*
3. It makes a name component
4. It makes a path
5. It gets a reference to the CORBA object called "ShapeList", using *resolve* and narrows it

it invokes the *allSha* containing remote re stored by the server

it uses one of the remote references in the array to invoke the *getAllState* method in the corresponding CORBA object whose type is *Shape*
the value returned is of type *GraphicalObject*

Outline

- Common Object Request Broker Architecture (CORBA)
- Distributed Common Object Model (DCOM)

Distributed Component Object Model (DCOM)

- Designed by Microsoft
- Based on Component Object Model (COM)
- Addresses issues such as:
  - Interoperability
    - Different applications, platforms, languages
  - Versioning
    - Compatibility between a new version of a server and old versions of clients
      - New interfaces should preserve the old interface
  - Naming
    - Use Globally unique identifiers

History

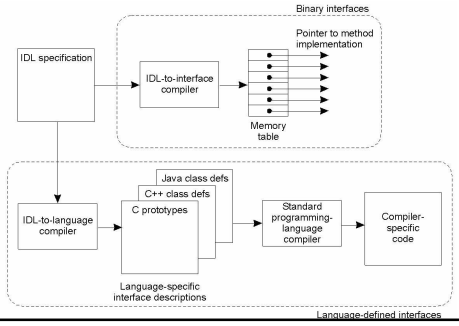
- DDE → OLE1 → COM → OLE → DCOM
- Dynamic Data Exchange (DDE)
  - For data exchange between any application through clipboard package
  - Originally for Windows 2.1
- Object Linking and Embedding (OLE v1.0)
  - A compound document can *embed* objects belonging to other applications
  - E.g., an Excel spreadsheet in a Word document
  - An *embedded object* is linked to its original application
  - Restricted to document objects

History (continued)

- Component Object Model (COM)
  - Interoperability of components
  - Ability to share non-document based components
  - Object-based technology
    - Identity, polymorphism (multiple interfaces to a component), interface inheritance
- OLE
  - Layered on top of COM (and DCOM)
  - Links the application layer to the underlying COM architecture

Object Model

- The difference between language-defined (CORBA) and binary interfaces (DCOM)



DCOM Properties

- Distributed shared memory management
  - DCOM provides interfaces for distributed components to share memory
- Network interoperability and transparency
- Dynamic loading and unloading
  - DCOM manages reference counts to objects
  - Unloads objects whose reference count is 0
- Status reporting
  - Of remote execution using HRESULT struct

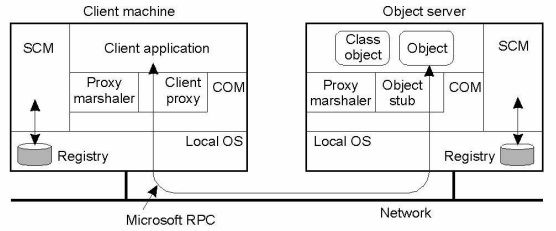
37

DCOM Services

- DCOM is responsible for initializing a connection between components, and
  - Negotiating protocols for communication
- DCOM provides support for persistent storage
  - Objects can persist
- Components can be assigned “intelligent names” called monikers

38

DCOM Architecture



- SCM: Service Control Manager

39

Creating objects

- Classes of objects have globally unique identifiers (GUIDs)
  - 128 bit numbers
  - Also called class ids (CLSID)
- DCOM provides functions to create objects given a server name and a class id
  - The SCM on the client connects to the SCM of the server and requests creation of the object

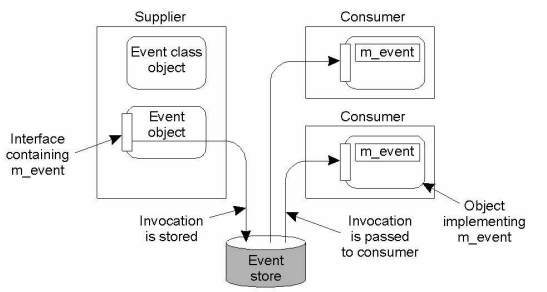
40

MIDL

- An extension of DCE’s IDL
- The MIDL compiler generates the client and server stub files
- Every DCOM interface inherits from an interface known as IUnknown
  - Interface names start with I
  - IUnknown has three methods
    - AddRef(), Release() and QueryInterface()
    - AddRef() and Release() are used to manage reference counts (for memory management)

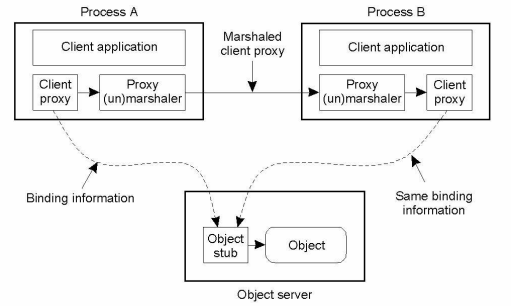
41

Events



42

Passing an Object Reference in DCOM (with custom marshaling)



Monikers (1)

- Object names (as opposed to class names) are called monikers
- A moniker distinguishes one instance from another of the same class
- Monikers themselves are objects
- A moniker carries enough information to locate the object it represents
  - They can also recreate the object, if it is not currently running
- They have a human readable form similar to a URL. Example: Moniker for a file object "`file:c:\my documents\July Report.doc`"

Monikers (2)

- When a client passes a moniker to access an object, COM looks up a Running Object Table (ROT) for the moniker name
  - If it exists, a pointer to the object is returned
  - Else, a new object instance is created, its state is restored, its reference is entered in ROT, and a pointer to the object is returned to the client
    - Monikers contain reference to the object's persisted state

Fault Tolerance

- Supported by mean of transactions
- Developer specify that a series of method invocations should be grouped in a transaction

Attribute 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
DISABLED	Never join a transaction, even if told to do so

Declarative Security (1)

- Authentication levels in DCOM

Authentication 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

Declarative Security (2)

- Impersonation levels in DCOM

Impersonation level	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

Programmatic Security (1)

- Allow applications to security levels, and choose between different security services
- Default authentication services supported in DCOM:

Service	Description
NONE	No authentication
DCE_PRIVATE	DCE authentication based on shared keys
DCE_PUBLIC	DEC authentication based on public keys
WINNT	Windows NT security
GSS_KERBEROS	Kerberos authentication

49

Programmatic Security (2)

Default authorization services supported in DCOM

Service	Description
NONE	No authorization
NAME	Authorization based on the client's identity
DCE	Authorization using DEC Privilege Attribute Certificates (PACs)

50

CORBA vs. DCOM (1)

Issue	CORBA	DCOM
Design goals	Interoperability	Functionality
Object model	Remote objects	Remote objects
Services	Many of its own	From environment
Interfaces	IDL based	Binary
Sync. communication	Yes	Yes
Async. communication	Yes	Yes
Callbacks	Yes	Yes
Events	Yes	Yes
Messaging	Yes	Yes
Object server	Flexible (POA)	Hard-coded
Directory service	Yes	Yes
Trading service	yes	No

51

CORBA vs. DCOM (2)

Issue	CORBA	DCOM
Naming service	Yes	Yes
Location service	No	No
Object reference	Object's location	Interface pointer
Synchronization	Transactions	Transactions
Replication support	Separate server	None
Transactions	Yes	Yes
Fault tolerance	By replication	By transactions
Recovery support	Yes	By transactions
Security	Various mechanisms	Various mechanisms

52