CprE 450/550x
Distributed Systems and Middleware

Distributed Object-based Systems CORBA

Yong Guan
3216 Coover
Tel: (515) 294-8378
Email: guan@ee.iastate.edu

March 30, 2004

Readings for Today's Lecture

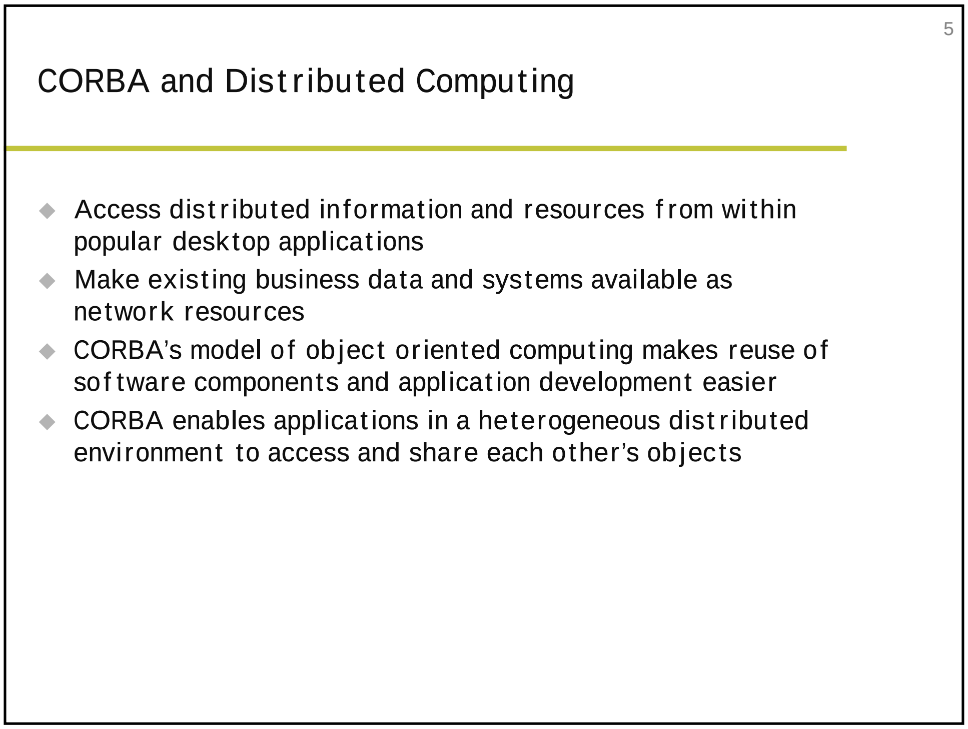
- References
  - Chapter 9 of "Distributed Systems: Principles and Paradigms"
  - http://www.corba.org/
  - http://www.omg.org/gettingstarted/
  - http://www.omg.org/gettingstarted/readingroom.htm
  - "Understanding CORBA"
  - "Examples of Writing CORBA Applications", http://www.cs.wustl.edu/~schmidt/PDF/corba-apps4.pdf
  - "Introduction to Distributed Object Programming with CORBA ", http://www.cs.wustl.edu/~schmidt/PDF/corba4.pdf

Outline

- Role of CORBA and need for object oriented distributed computing
- A simple CORBA architecture
- CORBA client-server example
- Coding with IDL
- Complete CORBA architecture and its various components
- Some CORBA products and vendors

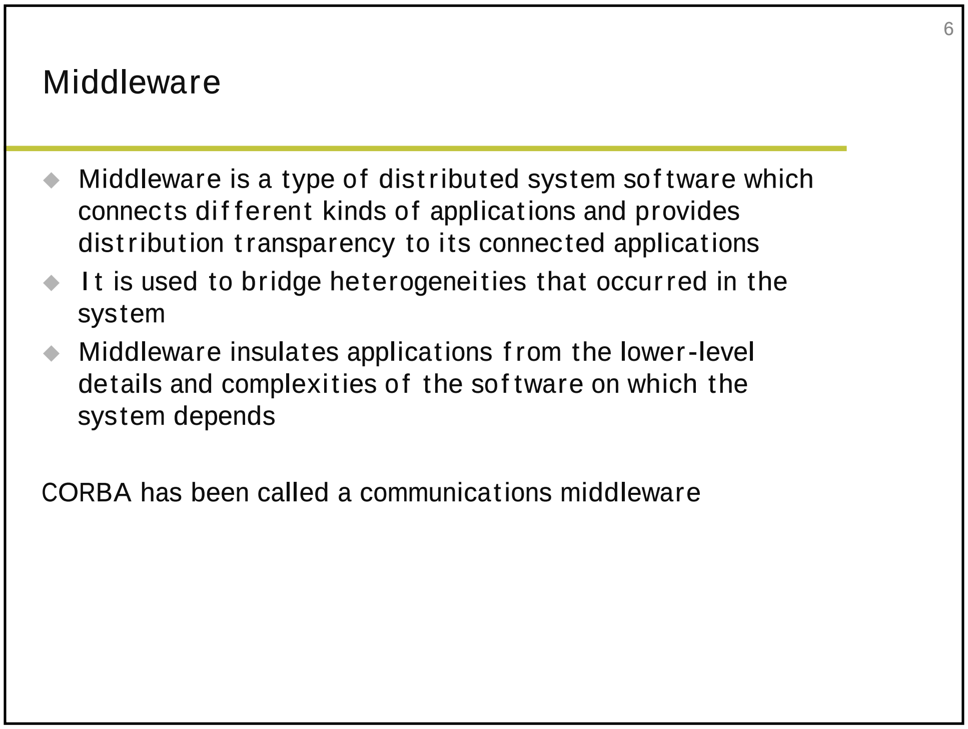
CORBA and OMG

- CORBA (Common Object Request Broker Architecture) is a **standard** for distributed objects being developed by the Object Management Group (OMG) that provides the mechanisms by which objects transparently make requests and receive responses
- CORBA provides interoperability between applications built in (possibly) different languages, running on (possibly) different machines in heterogeneous distributed environments
- The OMG is a consortium of software vendors and end users



CORBA and Distributed Computing

- Access distributed information and resources from within popular desktop applications
- Make existing business data and systems available as network resources
- CORBA's model of object oriented computing makes reuse of software components and application development easier
- CORBA enables applications in a heterogeneous distributed environment to access and share each other's objects

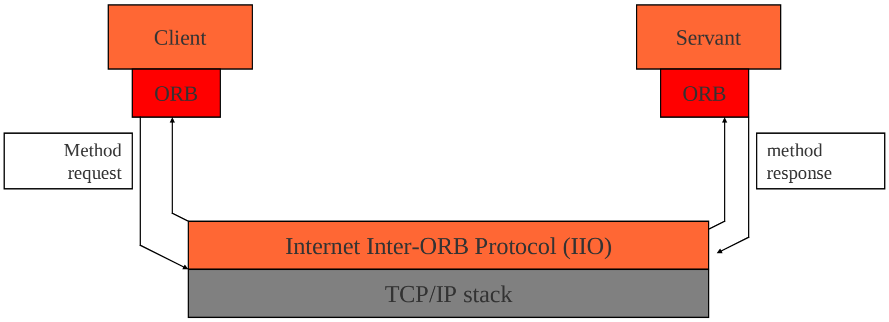


Middleware

- Middleware is a type of distributed system software which connects different kinds of applications and provides distribution transparency to its connected applications
- It is used to bridge heterogeneities that occurred in the system
- Middleware insulates applications from the lower-level details and complexities of the software on which the system depends

CORBA has been called a communications middleware

Simple CORBA Architecture



ORB (Object Request Broker)

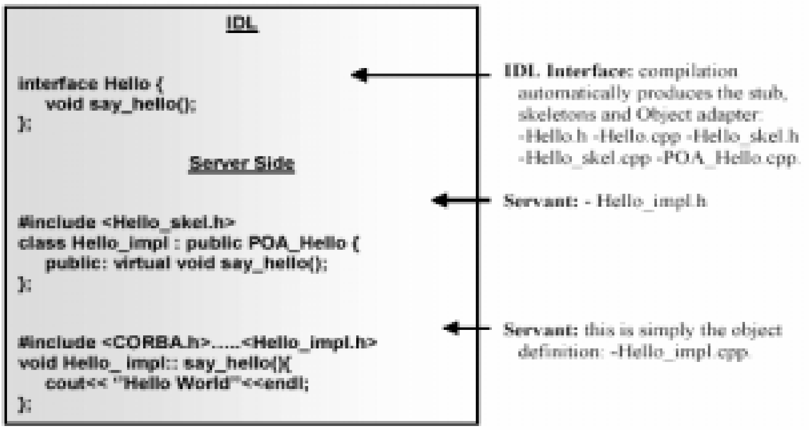
- Uses Object Reference to identify and locate objects

  ***Object Reference*: A handle to an object that a client must hold in order to access the object**
- Delivers request to objects
- Returns output values back to client
- Services necessary to accomplish the tasks are completely transparent to the client

CORBA Application Development

- Steps in developing a CORBA server and client
  - Design your application interface and specify them in OMG IDL (Interface Definition Language)
  - Run the IDL specs through IDL compiler of language of your choice, say C++, to generate client-side stub and server-side skeleton
  - Implement server side interfaces using C++ classes (called servants)
  - Implement the server program that instantiates the servants
  - Compile the server program along with the skeleton code using a C++ compiler
  - Implement the client program
  - Compile the client program along with the stub code

A Sample Client-Server Program



IDL

```
interface Hello {
    void say_hello();
};
```

Server Side

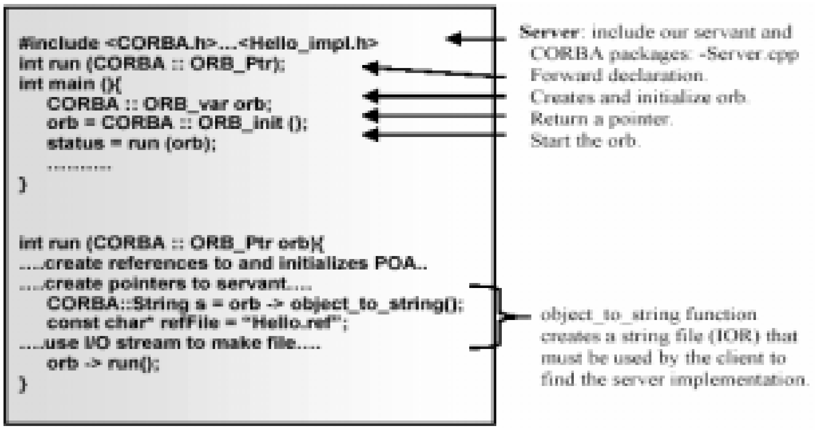
```
#include <Hello_skel.h>
class Hello_impl : public POA_Hello {
    public: virtual void say_hello();
};

#include <CORBA.h>.....<Hello_impl.h>
void Hello_ impl:: say_hello(){
    cout<< "Hello World"<<endl;
};
```

IDL Interface: compilation automatically produces the stub, skeletons and Object adapter: -Hello.h -Hello.cpp -Hello_skel.h -Hello_skel.cpp -POA_Hello.cpp.

Servant: - Hello_impl.h

Servant: this is simply the object definition: -Hello_impl.cpp.

Client-Server Program (Cont.)



```
#include <CORBA.h>....<Hello_impl.h>
int run (CORBA :: ORB_Ptr);
int main (){
    CORBA :: ORB_var orb;
    orb = CORBA :: ORB_init ();
    status = run (orb);
    ..........
}

int run (CORBA :: ORB_Ptr orb){
....create references to and initializes POA..
....create pointers to servant....
    CORBA::String s = orb -> object_to_string();
    const char* refFile = "Hello.ref";
....use I/O stream to make file....
    orb -> run();
}
```

Server: include our servant and CORBA packages: -Server.cpp

Forward declaration.

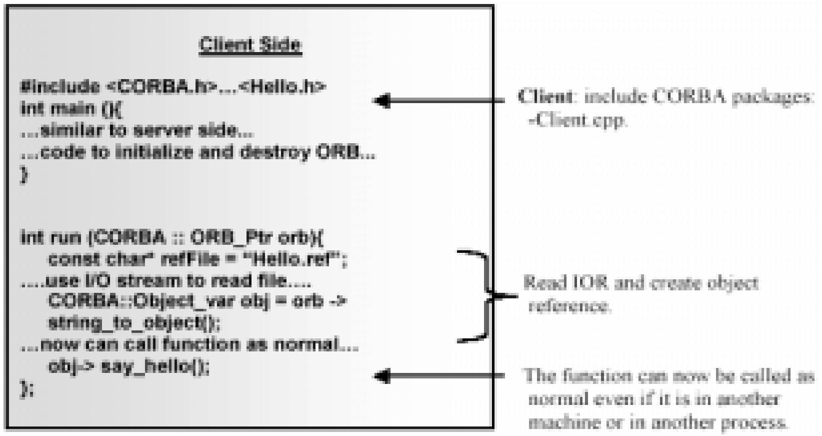
Creates and initialize orb.

Return a pointer.

Start the orb.

object_to_string function creates a string file (IOR) that must be used by the client to find the server implementation.

Client-Server Program (Cont.)



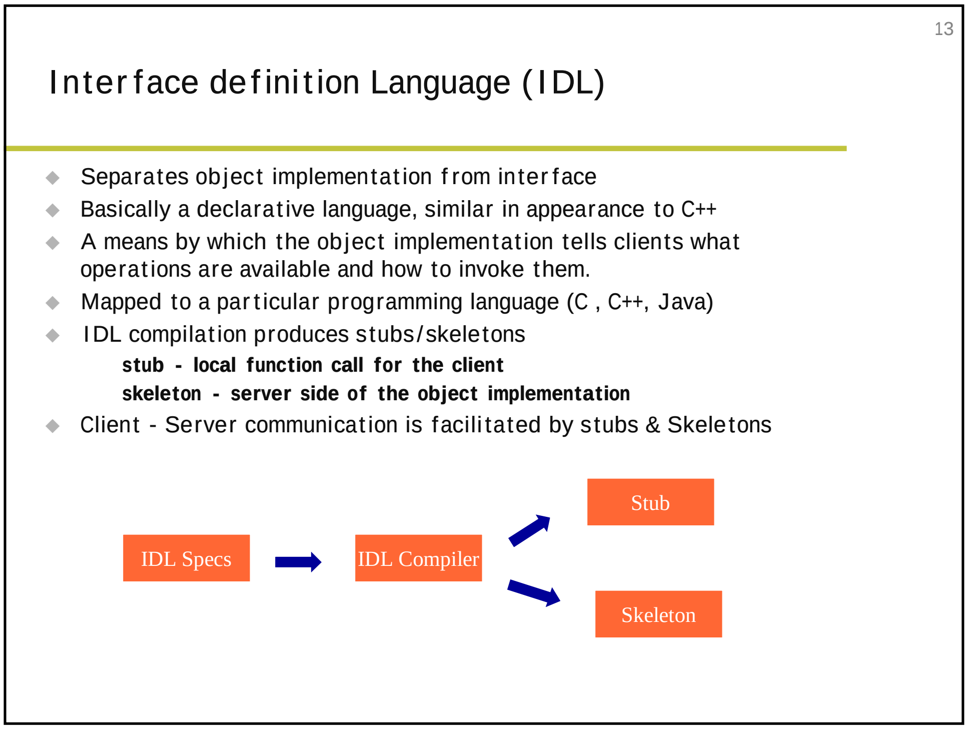
Client Side

```
#include <CORBA.h>....<Hello.h>
int main (){
....similar to server side...
....code to initialize and destroy ORB...
}

int run (CORBA :: ORB_Ptr orb){
    const char* refFile = "Hello.ref";
.....use I/O stream to read file....
    CORBA::Object_var obj = orb ->
    string_to_object();
....now can call function as normal...
    obj-> say_hello();
};
```

Client: include CORBA packages: -Client.cpp.

Read IOR and create object reference.

The function can now be called as normal even if it is in another machine or in another process.

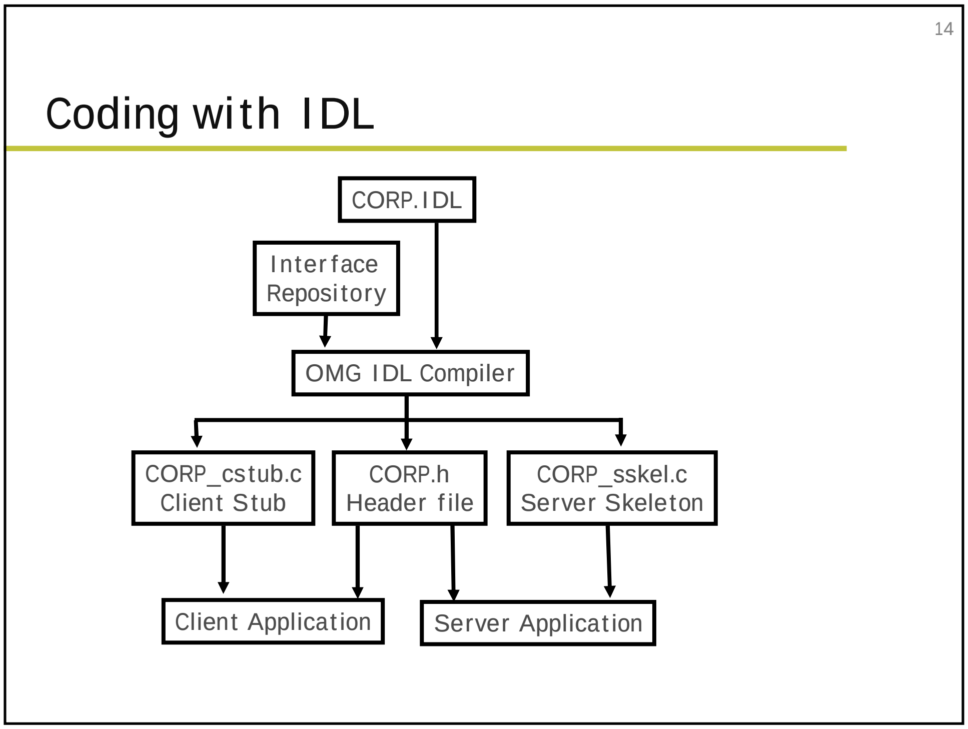


13

Interface definition Language (IDL)

- Separates object implementation from interface
- Basically a declarative language, similar in appearance to C++
- A means by which the object implementation tells clients what operations are available and how to invoke them.
- Mapped to a particular programming language (C , C++, Java)
- IDL compilation produces stubs/skeletons
  - **stub - local function call for the client**
  - **skeleton - server side of the object implementation**
- Client - Server communication is facilitated by stubs & Skeletons

IDL Specs → IDL Compiler → Stub, Skeleton



14

Coding with IDL

CORP.IDL

Interface Repository

OMG IDL Compiler

CORP_cstub.c Client Stub

CORP.h Header file

CORP_sskel.c Server Skeleton

Client Application

Server Application

Coding with IDL (cont.)

```
//File CORP.IDL

Module CORP
{
      typedef long BadgeNum;
      typedef long DeptNum;
      enum DismissalCode {DISMISS_FIRED, DISMISS_QUIT}

      Interface Employee
      {
               void promote(in char newjobclass);
               void dismiss(in DismissalCode reason,
                                 in string description);
      }

      ……
}
```

Coding with IDL (cont.)

```
//File CORP.IDL    ---- Defining an object attribute in ODL

Module CORP
{
      typedef long BadgeNum;
      typedef long DeptNum;
      enum DismissalCode {DISMISS_FIRED, DISMISS_QUIT}

      struct DeptInfo
               {
                              DeptNum id;
                              string name;
               }

      Interface Department
      {
               atttribute DeptInfo DeptID;
      }

      ……
}
```

17

Coding with IDL (cont.)

```
//File CORP.IDL    ---- Defining an read-only object attribute in ODL

Module CORP
{
Interface Employee;

     struct DeptInfo
              {
                             DeptNum id;
                             string name;
              }

     Interface Department
     {
              atttribute DeptInfo DeptID;
              readonly attribute Employee manager_obj;
     }
     Interface Employee
     {
              attribute EmpData personal_data;
              readonly attribute Department department_obj;
     }
     ……
}
```

18

Coding with IDL (cont.)

```
//File CORP.IDL    ---- Defining inheritance in ODL: single inheritance

Module CORP
{
     struct PersonalData    {
                             string lastname;
                             string firstname;
                             string phone;
              }
     typedef PersonalData EmpPersonalData;
     struct EmpData {
                             BadgeNum id;
                             char job_class;
                             float hourly_rate;
              }
     Interface Employee
     {
              attribute EmpData personal_data;
              readonly attribute Department department_obj;
              void promote(in char new_job_class);
              void dismiss(……);
              void transfer(……);
     }
     Interface Manager: Employee
     {
              void approve_transfer(……);
     }
     ……
}
```

Coding with IDL (cont.)

```
//File CORP.IDL    ---- Defining inheritance in ODL: multiple inheritance

Module CORP
{
      Interface Employee
      {
               ……
      }
      Interface Manager: Employee
      {
               ……
      }
      Interface Peronnel: Employee
      {
               ……
      }

      Interface PeronellManager: Personnel, Employee
      {
               ……
      }
      ……
}
```

Coding with IDL (cont.)

```
//File CORP.IDL    ---- Defining inheritance in ODL: inheritance across modules

Module CORP
{
      Interface PeronellManager: Personnel, Employee
      {
               ……
      }
      ……
}

Module ENGINEERING
{
      Interface EmployeeLocator
      {
               void FindEngineer(in CORP::BadgeNum id,
                                        out CORP::PersonalData info);
      }
      Interface PersonnelManager: CORP::PersonnelManger
      {
      }
}
```

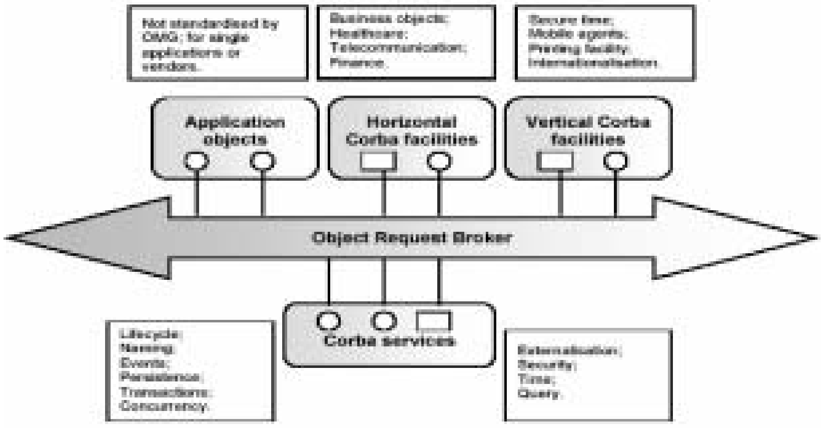
Coding with IDL (cont.)

```
//File CORP.IDL    ---- Defining User-defined Exceptions

Module CORP
{
     enum DenyApprovalReasons {REASON, CODES};
     exception DENY_APPROVAL
             {
                          DenyApprovalReasons reason;
             }
      Interface Manager: Employee
     {
             void approval_transfer (in Employee employee_obj,
                                     in Department current_department,
                                     in Department new_department)
                                     raises (DENY_APPROVAL);
     }
     ……
}
```

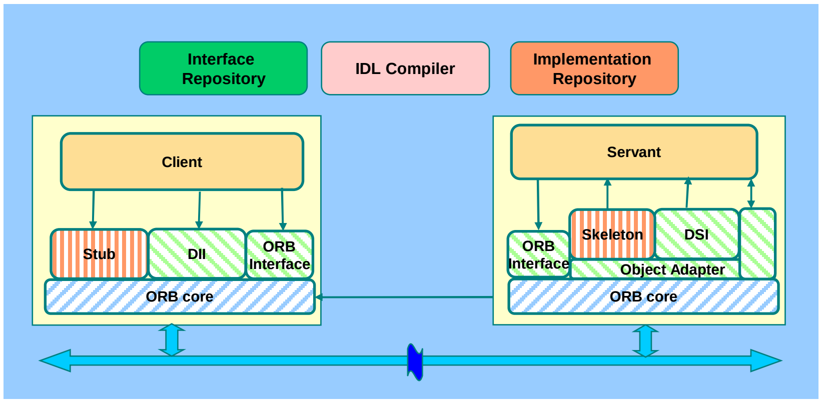
Coding with IDL (cont.)

```
//File CORP.IDL    ---- Defining context objects

Module CORP
{
      Interface Manager: Employee
     {
             void approval_transfer (in Employee employee_obj,
                                     in Department current_department,
                                     in Department new_department)
                                     raises (DENY_APPROVAL)
                                     context("division");
     }
     ……
}
```

The Object Management Architecture


Not standardised by OMG; for single applications or vendors.
Business objects; Healthcare; Telecommunication; Finance.
Secure time; Mobile agents; Printing facility; Internationalisation.
Application objects
Horizontal Corba facilities
Vertical Corba facilities
Object Request Broker
Corba services
Lifecycle; Naming; Events; Persistence; Transactions; Concurrency.
Externalisation; Security; Time; Query.

CORBA Components



Static and Dynamic Invocation Interface

- Static Invocation Interface (SII)
  - **Client knows interface operations in advance**
  - **Client is compiled with the relevant stub**
  - **During invocation, the proxy object understands the parameters in an operation and marshals them into the request**
- Dynamic Invocation Interface (DII)
  - **A client may not always have the stub available at compile time**
    - **Bridges, Proxy servers**
  - **Allows clients to discover operations parameters using Interface Repository and create requests dynamically**
  - **More flexible but less efficient. Also, more complicated and less typesafe**

Interface Repository (IFR)

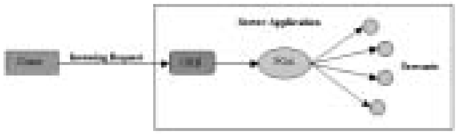
- A service that provides persistent objects that represent the IDL information in a form available at runtime
- Provides type information necessary to issue requests using the DII
- Also stores additional information like debugging info , libraries of stubs or skeletons etc

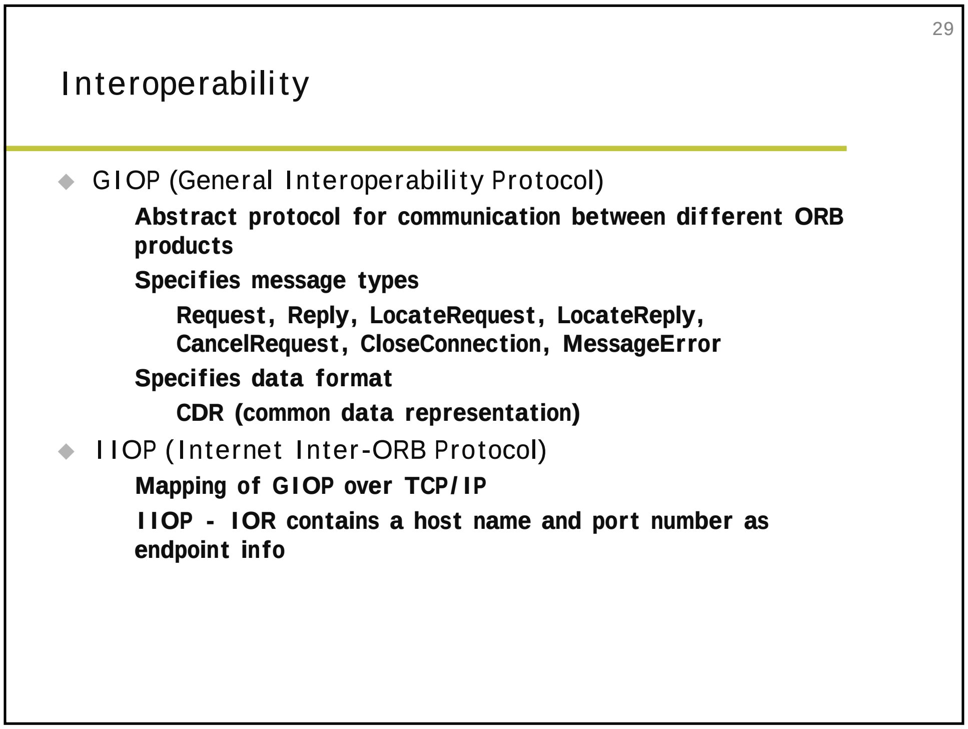
Static and Dynamic Skeleton Interface

- Static Skeleton Interface (SSI)
  - **Similar to SII, but on server side**
  - **Knows the operation types at compile time**
  - **Performs request demarshaling and dispatching**
- Dynamic Skeleton Interface (DSI)
  - **Similar to DII, but on server side**
  - **Generic skeleton interface for all objects**

Object Adaptor (OA)

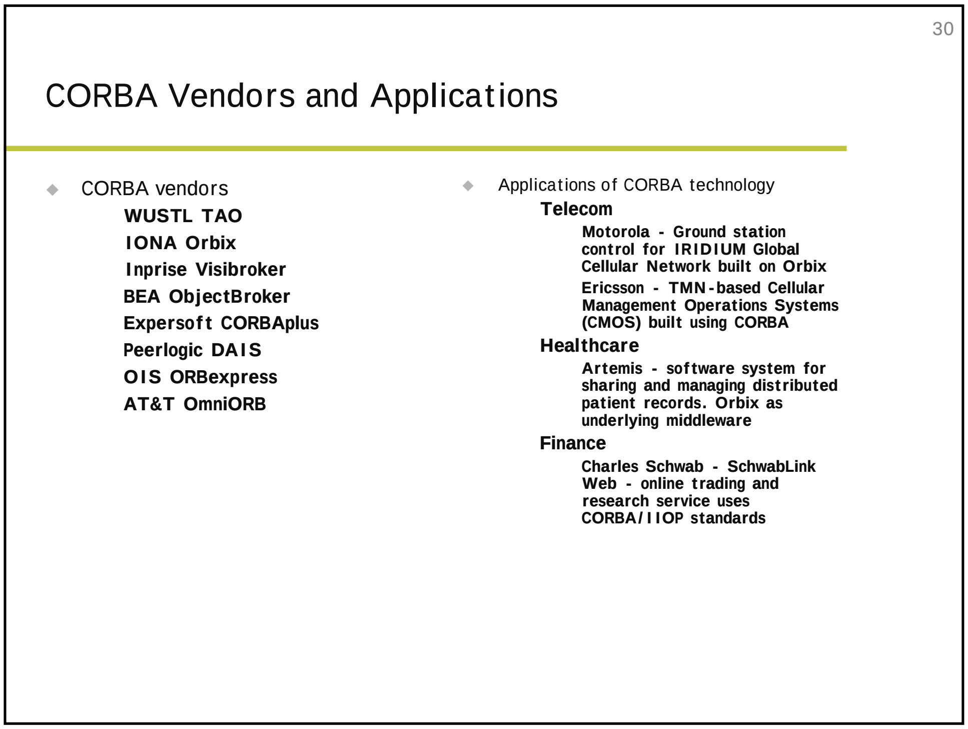
- Implementations must be registered with the OA
- When a client requests a service from an object, the OA maps the request to the appropriate implementation
- Activate and deactivate objects
- Objects can be implemented as C++ classes or C functions
- Allowing varied methods of implementation facilitates integration of legacy applications
- Two types – **BOA** (basic) and **POA (portable)**





Interoperability

- GIOP (General Interoperability Protocol)
  - **Abstract protocol for communication between different ORB products**
  - **Specifies message types**
    - **Request, Reply, LocateRequest, LocateReply, CancelRequest, CloseConnection, MessageError**
  - **Specifies data format**
    - **CDR (common data representation)**
- IIOP (Internet Inter-ORB Protocol)
  - **Mapping of GIOP over TCP/IP**
  - **IIOP - IOR contains a host name and port number as endpoint info**



CORBA Vendors and Applications

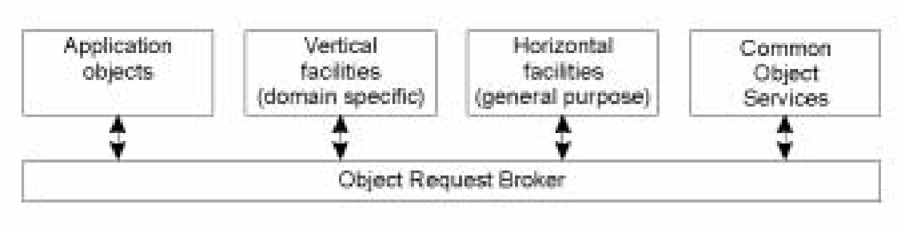
- CORBA vendors
  - **WUSTL TAO**
  - **IONA Orbix**
  - **Inprise Visibroker**
  - **BEA ObjectBroker**
  - **Expersoft CORBAplus**
  - **Peerlogic DAIS**
  - **OIS ORBexpress**
  - **AT&T OmniORB**

- Applications of CORBA technology
  - **Telecom**
    - **Motorola - Ground station control for IRIDIUM Global Cellular Network built on Orbix**
    - **Ericsson - TMN-based Cellular Management Operations Systems (CMOS) built using CORBA**
  - **Healthcare**
    - **Artemis - software system for sharing and managing distributed patient records. Orbix as underlying middleware**
  - **Finance**
    - **Charles Schwab - SchwabLink Web - online trading and research service uses CORBA/IIOP standards**

Any Questions?

See you next time.

Overview of CORBA



Application objects

Vertical facilities (domain specific)

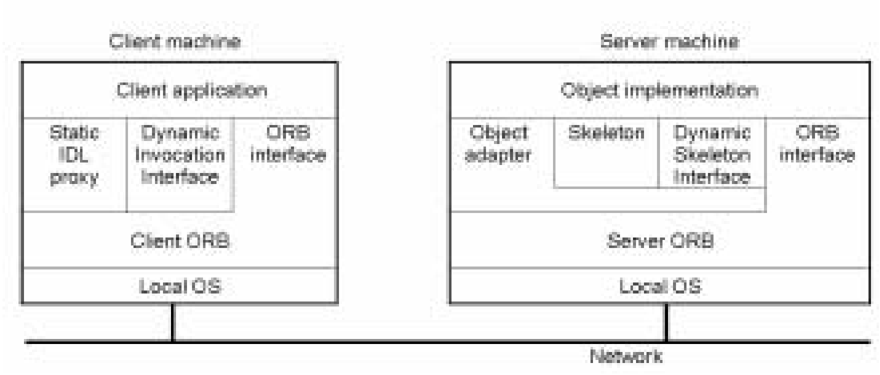
Horizontal facilities (general purpose)

Common Object Services

Object Request Broker

The global architecture of CORBA.

Object Model



The general organization of a CORBA system.

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 name of objects
Property	Facilities for associating (attribute, value) pairs with objects
Trading	Facilities to publish and find the services on 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

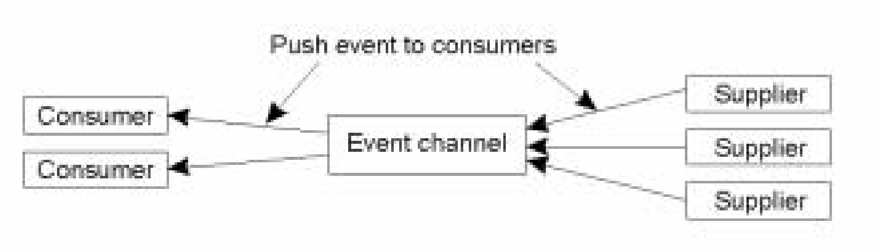
Overview of CORBA services.

Object Invocation Models

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

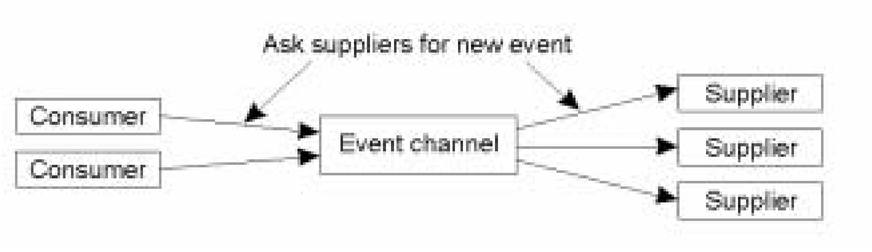
Invocation models supported in CORBA.

Event and Notification Services (1)



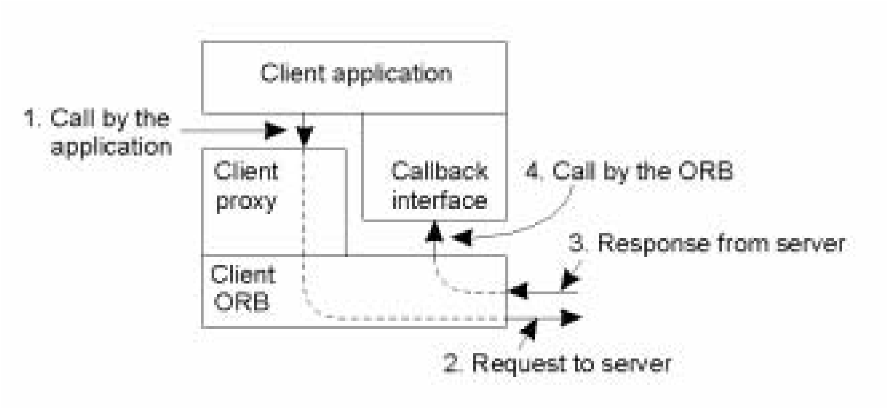
The logical organization of suppliers and consumers of events, following the push-style model.

Event and Notification Services (2)



The pull-style model for event delivery in CORBA.

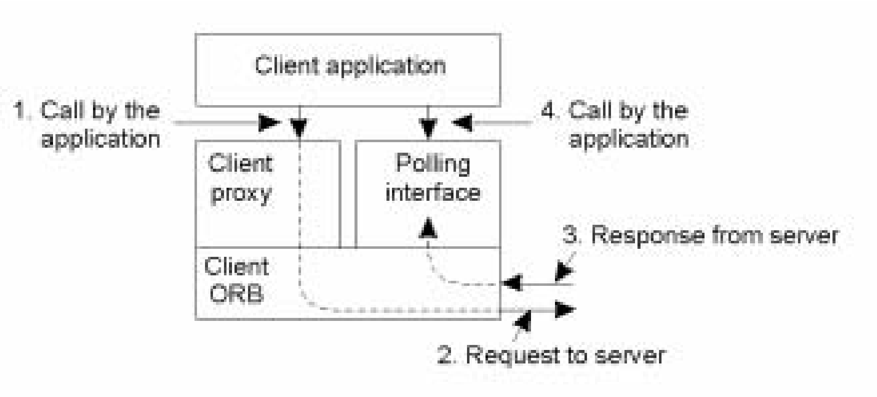
Messaging (1)



CORBA's callback model for asynchronous method invocation.

39

Messaging (2)



CORBA'S polling model for asynchronous method invocation.

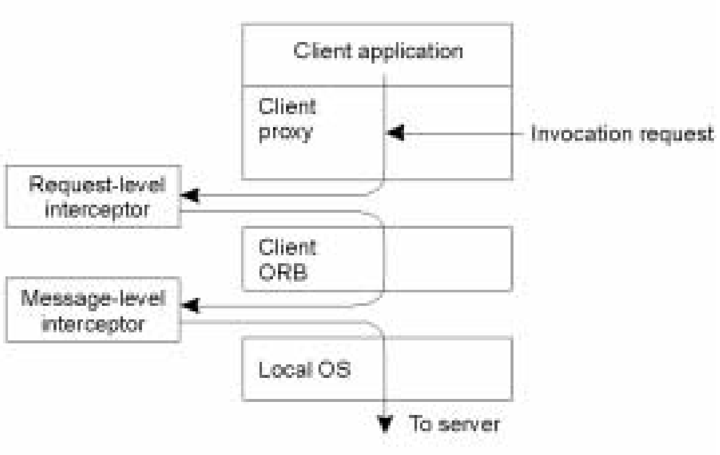
40

Interoperability

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

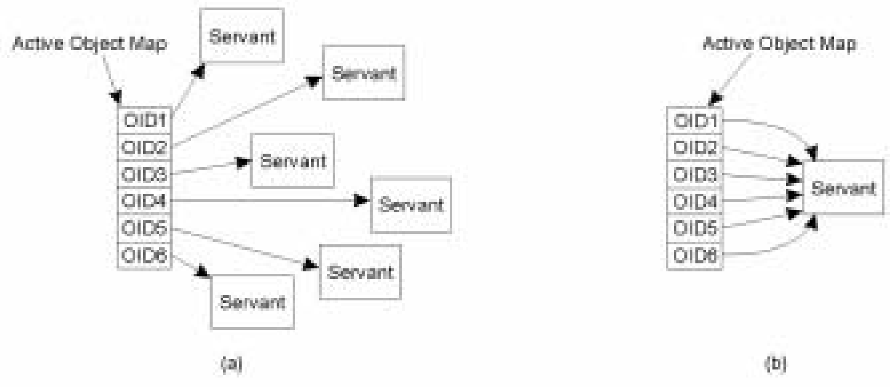
GIOP message types.

Clients



Logical placement of interceptors in CORBA.

Portable Object Adaptor (1)



Mapping of CORBA object identifiers to servants.

a) The POA supports multiple servants.

b) The POA supports a single servant.

43

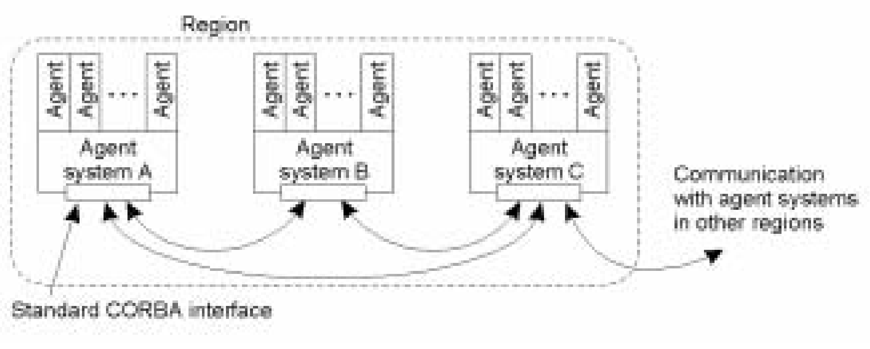
Portable Object Adaptor (2)

```
My_servant *my_object;              // Declare a reference to a C++ object
CORBA::Objectid_var oid;            // Declare a CORBA identifier
my_object = new MyServant;          // Create a new C++ object
oid = poa ->activate_object (my_object);
                                    // Register C++ object as CORBA OBJECT
```

Changing a C++ object into a CORBA object.

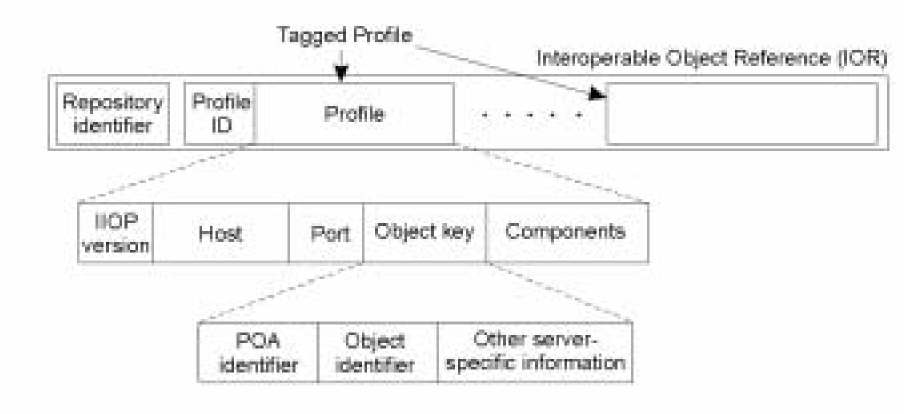
44

Agents



CORBA's overall model of agents, agent systems, and regions.

Object References (1)



Tagged Profile

Interoperable Object Reference (IOR)

Repository identifier

Profile ID

Profile

. . . . .

IIOP version

Host

Port

Object key

Components

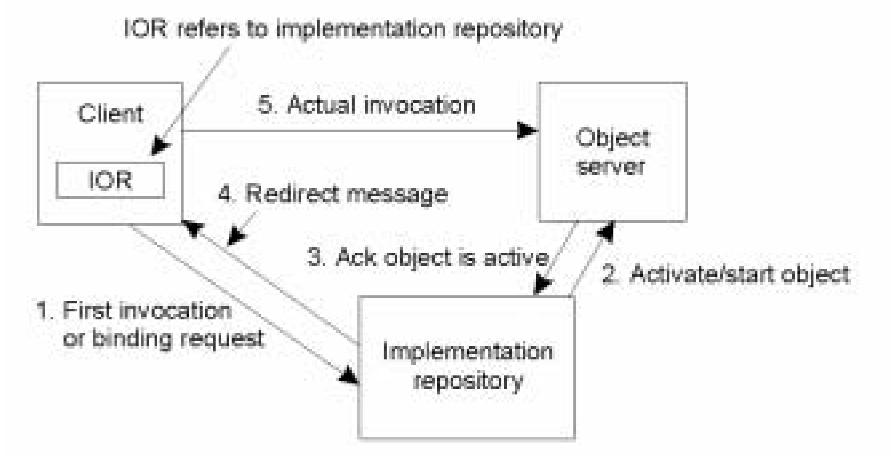
POA identifier

Object identifier

Other server-specific information

The organization of an IOR with specific information for IIOP.

Object References (2)



IOR refers to implementation repository

Client

IOR

5. Actual invocation

Object server

4. Redirect message

3. Ack object is active

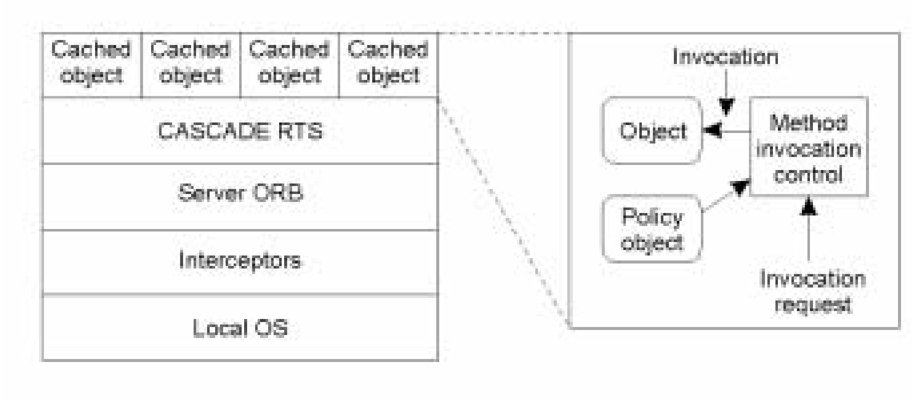
2. Activate/start object

1. First invocation or binding request

Implementation repository

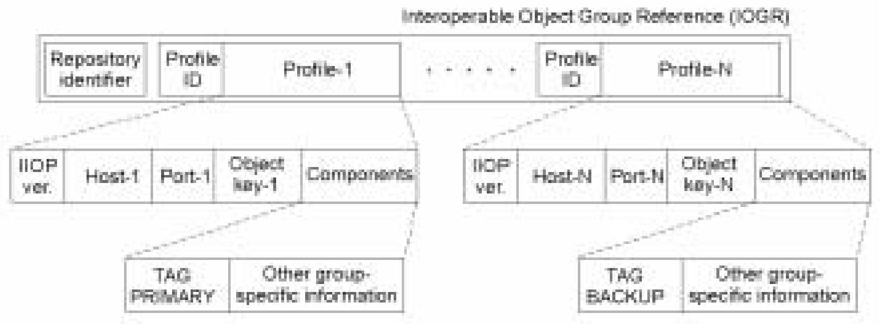
Indirect binding in CORBA.

Caching and Replication



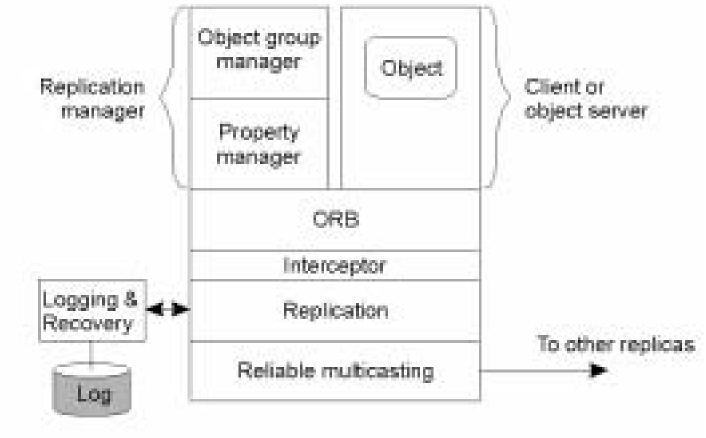
The (simplified) organization of a DCS.

Object Groups



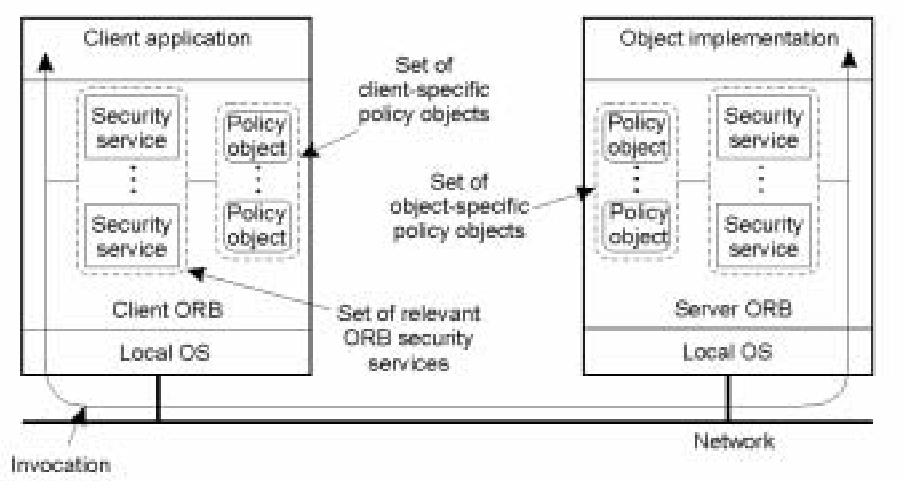
A possible organization of an IOGR for an object group having a primary and backups.

An Example Architecture



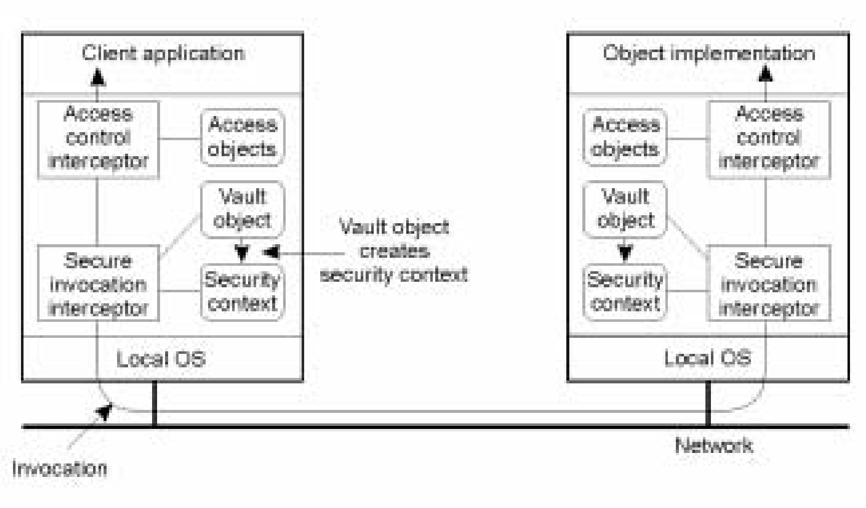
An example architecture of a fault-tolerant CORBA system.

Security (1)



The general organization for secure object invocation in CORBA.

Security (2)



The role of security interceptors in CORBA.

Any Questions?

See you next time.