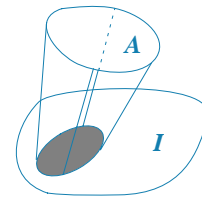
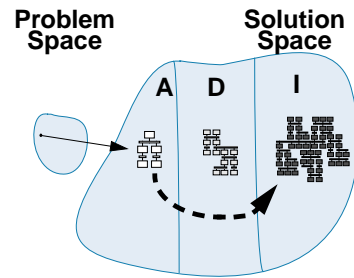


## Review — From Architecture to Implementation

- Directly implementing an architecture is infeasible
  - reduces to transformational programming
- Possible by limiting the target space
  - middleware platforms
  - software bus technologies



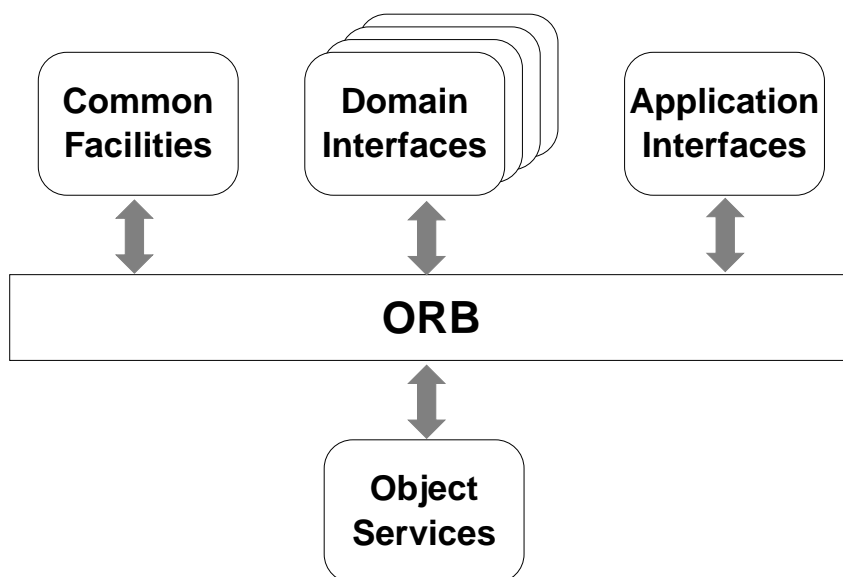
## Review — What is Middleware?

- Infrastructure that supports (distributed) component-based application development
  - a.k.a. distributed component platforms
  - mechanisms to enable component communication
  - mechanisms to hide distribution information
  - (large) set of predefined components
- *Software that resides between an application and the inner workings of the system hosting the application*
- Role of middleware in software architectures
  - architecture-based development is top-down
  - component-based development is bottom-up
  - middleware imposes certain architectural constraints
  - architecture can impose constraints on middleware

## What is CORBA?

- A middleware platform that supports a standardized OO architecture for software applications
  - **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
  - developed by OMG
- CORBA supports distributed object computing
  - distributed computing + OO computing
- It uses a *broker*
  - an intermediary handling requests in a system
  - facilitates communication between clients and server objects
  - separates a component's interface from its implementation
  - flexible system composition and evolution
- CORBA is a component of OMG's OMA
  - **O**bject **M**anagement **A**rchitecture

## Overview of OMA



## OMA Object Interface Categories

- Object services
    - fundamental services
    - e.g., object location (naming, trading)
  - Common facilities
    - oriented toward end-user applications
    - e.g., distributed document component facility
  - Domain interfaces
    - oriented towards a particular application domain
    - e.g., medical, telecommunication
  - Application interfaces
    - not standardized by CORBA
- *Object frameworks are interacting objects providing certain functionality in a given domain*

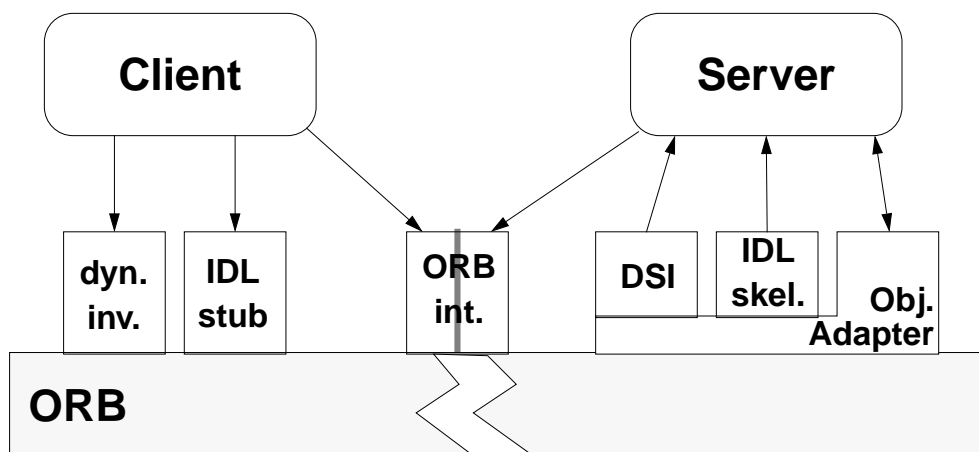
## CORBA's Enhancements to Client/Server

- Mutable client-server relationships
  - clients and servers not hard-wired to one another
- Interaction intermediary
  - ORB
  - object adapters
- Multi-process servers
- Both synchronous and deferred synchronous communication
  - deferred synchronous  $\approx$  asynchronous

## Main CORBA Features

- Object request broker (ORB)
- OMG interface definition language (IDL)
- Language mappings
- Stubs and skeletons
- Interface repository
- Dynamic invocation and dispatch
- Object adapters
- Inter-ORB protocols

## (Simplified) CORBA Architecture



## Object Request Broker

- Performs delivery of client requests and server responses
  - provides a layer of transparency between clients and servers
- The ORB hides object
  - location
  - implementation
  - execution state
  - communication mechanisms
- Requests on an object are made using its reference
- Clients can obtain references in three ways
  - create an object to get its reference
    - uses *factory objects*
  - invoke a lookup service (naming, trading)
  - convert to string and store for future reference

## OMG IDL

- Specifies (implementation-independent) object interface
  - all interfaces (implicitly) inherit from the *Object* interface
- Built-in types
  - short, long, long long (signed and unsigned)
  - float, double, long double
  - char, wchar
  - boolean, octet, enum
- Constructed types
  - struct
  - union
- Template types
  - exact structure at declaration time
  - string, wstring, sequence, fixed

## Example IDL Interface Definition

```
interface Employee
{
    void promote (in char          new_job_class);
    void dismiss (in DismissalCode reason,
                 in String         description);
};
```

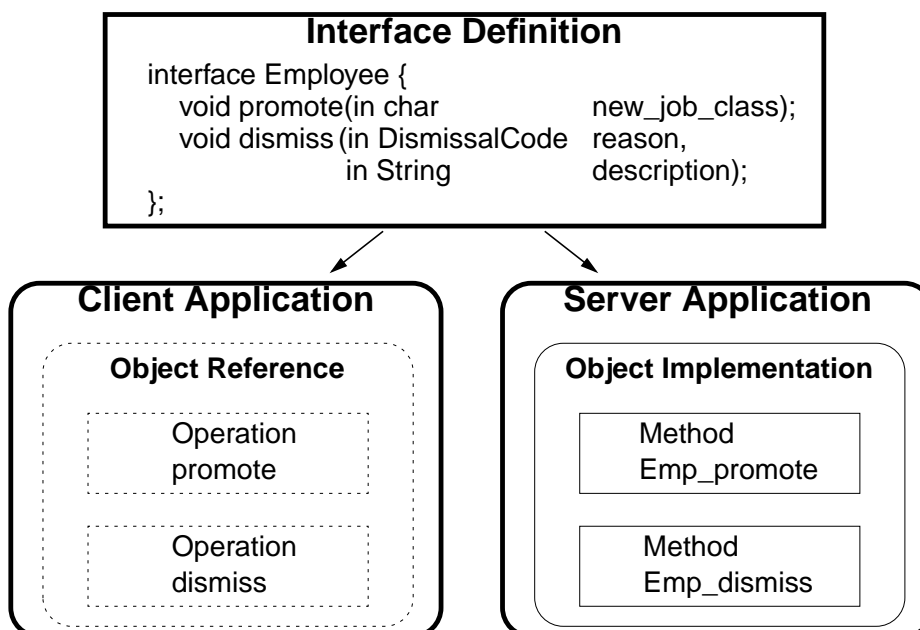
## Language Mappings

- IDL is language-independent
- Standardized mappings
  - C, C++, Ada95, Smalltalk, Java
- Additional mappings available, but not standardized
  - Perl, Eiffel, Modula-3
- CORBA's equivalent of Q's type concordance
  - marshalling
  - de-marshalling
  - platform-independent representation

## Stubs and Skeletons

- Used in CORBA's static invocation
- PL-specific counterparts to IDL definitions
- **Stubs** create and issue requests on the client side
  - a.k.a. surrogates or proxies
  - perform marshalling of requests
- **Skeletons** receive and forward requests to objects on the server side
  - perform unmarshalling of requests
  - returns results via the server and client ORBs to the stub

## Example of Stubs and Skeletons



## Interface Repository

- Used for accessing objects whose interface is not known at compile time
- Knowing interfaces of all objects *a priori* may be impractical
  - independently developed components
  - fast changing parts of the system
  - dynamic manipulation
- IR allows access to the IDL type system at runtime
  - IR is a CORBA object whose functionality is invoked like any other object
  - allows CORBA's dynamic invocation

## Dynamic Invocation and Dispatch

- Two supported interfaces
  - dynamic invocation interface (DII)
  - dynamic skeleton interface (DSI)
- DII
  - supports dynamic requests on the client side
  - generic stub
- DSI
  - supports dynamic request dispatch to objects on the server side
  - generic skeleton

## Object Adapters

- Serve as the glue between an object and the ORB
  - based on the adapter pattern
  - no need for direct attachments between objects and ORBs
  - a role associated with connectors
- OA responsibilities
  - object registration
  - object reference generation
  - server process activation
  - object activation
  - request demultiplexing
  - object upcalls — request dispatching to registered objects
- Different OA needed for each PL
  - only the Basic Object Adapter (BOA) currently provided

## Inter-ORB Protocols

- CORBA standard does not cover all of CORBA
  - e.g., different protocols and object references are possible
  - different vendors may provide incompatible support
- General ORB interoperability architecture
  - based on the General Inter-ORB Protocol (GIOP)
  - one instance of GIOP is the IIOP (built on top of TCP/IP)
  - Environment-Specific Inter-ORB Protocols (ESIOP)
    - allow CORBA and non-CORBA components to interact
    - allow CORBA to exploit existing support
  - one instance is the Distributed Computing Environment Common Inter-ORB Protocol (DSE-CIOP)
- Standard object reference format
  - Interoperable Object Reference (IOR)

## CORBA Deficiencies

- Difficult to optimize the ORB infrastructure to exploit known operational regularities
- Marshalling
  - complex marshalling even in a homogeneous case
- Transport
  - no alternative transports (e.g., shared memory)
  - can do so via, e.g., DCE-CIOP
- Application footprint is large
- Deadlock detection not supported
- Reclamation of objects (garbage collection) must be addressed

## CORBA Deficiencies (2)

- Local vs. remote transparency is not always good
- Remoteness cannot be hidden w.r.t. time and reliability
  - time — cannot hide the fact that remote access is slower than local access
  - reliability — higher potential for failure requires frequent insertion of exception handlers
- Programs are forced to treat all objects as (potentially) remote
  - may result in awkward coding style and awkward code

## CORBA Deficiencies (3)

- CORBA's assumptions about object naming are not scalable
  - all object names are kept in a global namespace
  - all objects are represented in clients (perhaps multiple times) via object references
- Problem for very large object-based applications
  - objects may contain millions of other objects
  - e.g., OODBMS
- Mechanisms may be needed to refer to arbitrarily nested objects
  - without globally naming them
  - without requiring surrogates