Università degli Studi di Bologna
Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Scuola di Ingegneria

Corso di

Reti di Calcolatori M

MIDDLEWARE - CORBA

Antonio Corradi

Anno accademico 2014/2015

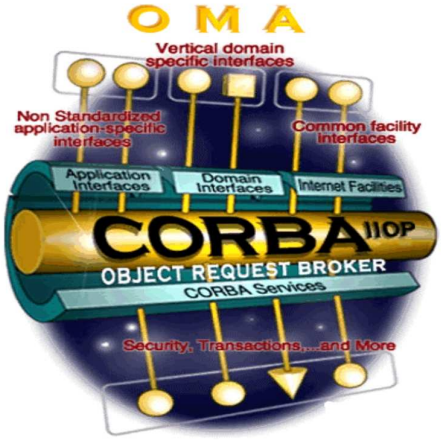
MIDDLEWARE: CORBA

OMG- Object Management Group

CORBA creato nel 1989 con **440 aziende** Microsoft, Digital, HP, NCR, SUN, OSF, *etc.* con l'obiettivo di creare un **sistema** di **uso** e **gestione** di una **architettura distribuita**

Common **O**bject **R**equest **B**roker **A**rchitecture
CORBA standard v1 ⇨ 1991, v1.2 ⇨ 1992
v2 ⇨ 1996, v3 ⇨ 2000
Orbix *SunOS Solaris, Iris, Windows NT, HP/UX, AIX, OSF/1, UnixWare*
DSOM *IBM*

Specifica di un middleware generale ad oggetti (componenti) per sistemi distribuiti eterogenei non legato ad un linguaggio specifico



MIDDLEWARE: CORBA

STANDARD SISTEMI APERTI AD OGGETTI

modelli ad oggetti eterogenei per consentire la **integrazione** e **interazione reciproca completa** tra oggetti **in compresenza** per **ambienti realizzati anche non ad oggetti (modello C/S)**

CORBA prevede

- definizione di un **linguaggio di interfaccia per i servizi**
- definizione e supporto della **interazione** tra **oggetti**
- **bus** per **integrazione** di oggetti **di ambienti diversi (ORB)**
- **interazione** anche tra sistemi diversi con diversi gestori
- **linguaggi diversi** di deployment (**language mapping**)

Obiettivo è consentire di **supportare i servizi** senza **limiti al ciclo di vita** a tutte le applicazioni utente

ARCHITETTURA CORBA

Common **O**bject **R**equest **B**roker **A**rchitecture **CORBA**, come **ambiente comune, O**bject **M**anagement **A**rchitecture, per ambienti multiarchitettura e multilinguaggio con massima integrazione di sistemi legacy ed un progetto differenziato di server e clienti

Object Request Broker (ORB) è il **cuore** della **architettura** e **facilitatore di comunicazione** (**broker**) e consente i collegamenti in modo **statico** e **dinamico** (!?) tra le entità

ORB si comporta come abilitatore sempre presente e permette:

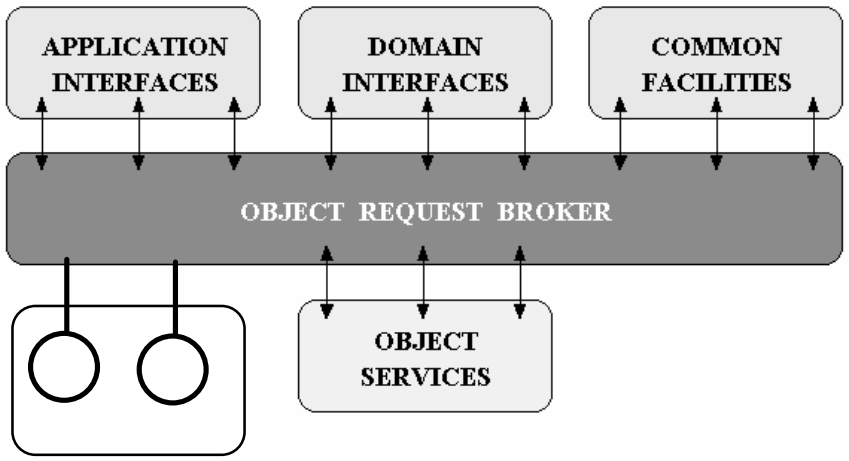
- controllo **allocazione** e **visibilità** di oggetti
- controllo dei **metodi** e della **comunicazione**
- controllo di **servizi accessori** disponibili sempre nella OMA per ogni language mapping
- **gestione facilitata** di tutti i possibili servizi

CORBA come middleware del terzo tipo, tempo di vita infinito

CORBA come BUS

ORB centro della **O**bject **M**anagement **A**rchitecture
ORB come un **bus centro di una architettura che permette di integrare tutte le risorse di una organizzazione**

Tutti gli oggetti applicativi gestiti possono appartenere ad **ambienti diversi** e devono potere **comunicare reciprocamente** senza necessità di **riprogetto**



Object Management Architecture

Altri componenti addizionali di ambiente

Common Facilities CF (orizzontali)

Insieme di funzionalità specifiche
Interfaccia utente (client-site),
Management Sistema, Informazioni, Task (server-site)

Domain Interfaces (verticali)

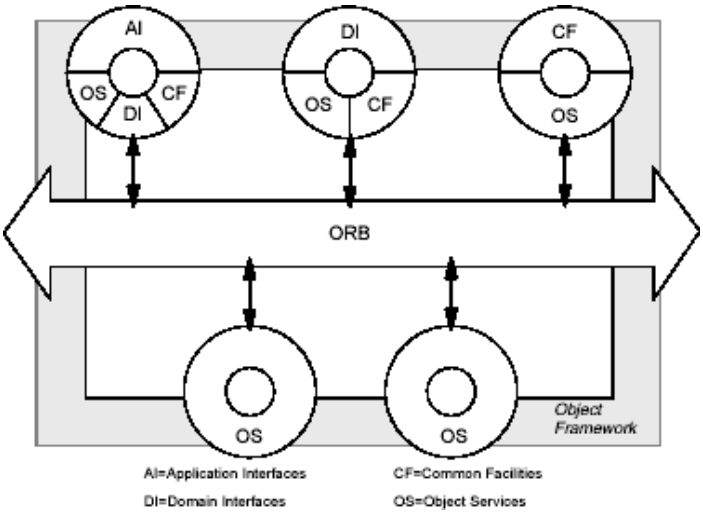
funzionalità dedicate ad aree applicative, ad es.
manifacturing, telecommunications, electronic commerce, transportation, business objects, healthcare, finance, life science, …

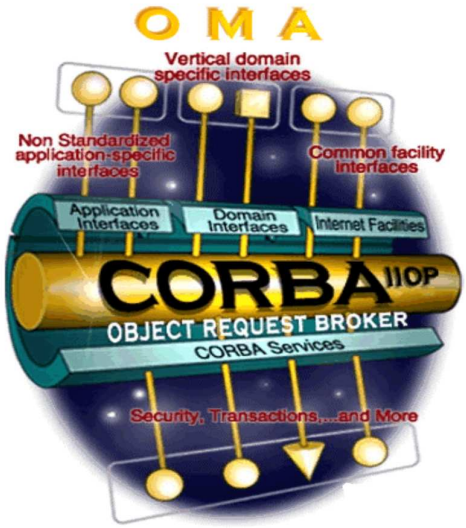
Application Interfaces

Non standardizzate in alcun modo e dipendenti dalla applicazione

Object Management Architecture - OMA

Ambiente Object Framework





Object Management Architecture

Ogni componente può legarsi ad ogni altro, preparando il tutto prima della esecuzione o **anche dinamicamente, se ignoto a priori,** usando il servizio di **uno o più ORB** (**noti dinamicamente**)

Insieme di **componenti addizionali di ambiente**

Object Services o CORBA Services (*Common Mw Services*)

Operazioni fondamentali per oggetti

- ***naming*** e ***trading*** service (compatibile con OO)
- ***event*** e ***notification*** service (meno Object-Oriented)

Oltre ad operazioni ulteriori (o servizi)

Per la gestione del tempo di vita, relazionali, transazioni, controllo concorrenza, sicurezza

COMPONENTI di CORBA

I componenti essenziali della architettura OMA, ossia di CORBA, associati ad un ORB:

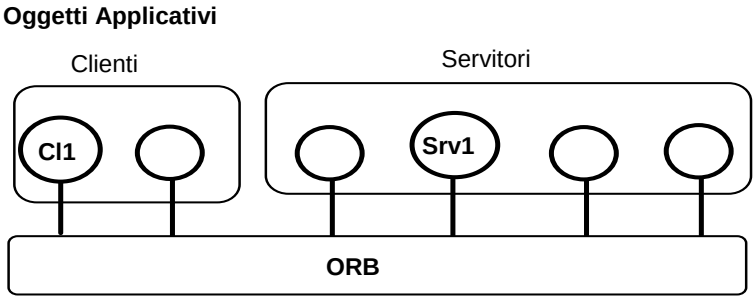
- **Object Request Broker** (ORB)
- **Interface Definition Language** (IDL)
- **Basic Object Adapter (e POA …)** (BOA e POA)
- **Static Invocation Interface** (SII)
- **Dynamic Invocation Interface** (DII)
- **Interface e Impl. Repository** (IR e IMR)
- **Protocolli per Integrazione** (GIOP)

Sono componenti a livello molto diverso

SUPPORTO CONTINUO dell'ORB

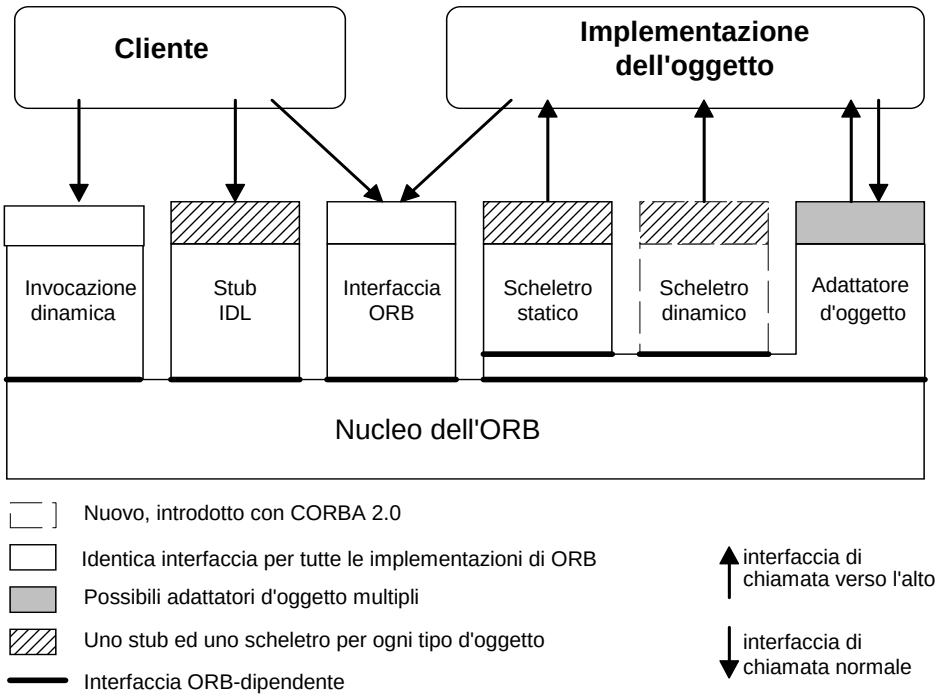
Object Request Broker (ORB) deve **coordinare la invocazione di servizi locali e remoti (in modo dinamico)**

- individuare **l'implementazione di un oggetto** come servitore ad una richiesta (object location)
- preparare il **servitore** a ricevere la richiesta - via *adattatore* (object creation, activation & management)
- trasferire **la richiesta** dal cliente al servitore
- restituire **la risposta** al cliente



CORBA: VISIONE DINAMICA

Elementi in gioco: visione complessiva utente



la visione è quella di CORBA 1.x

non cambiata fino a CORBA 3

LINGUAGGIO COMUNE in CORBA

Interface Definition Language (**CORBA IDL**) deve **coordinare e identificare i servizi richiesti e offerti, locali e remoti (per interazioni statiche o dinamiche)**

- Sia i **servitori** sia i **clienti** devono potere **identificarsi** e **rendersi noti** reciprocamente
- Sia le richieste di **operazioni** sia le offerte di **servizio** devono essere **associate al meglio**
- Si usa la esperienza **degli IDL** già sviluppati e diffusi per la definizione di un IDL generale multilinguaggio

Purtroppo IDL prevede una identificazione ed un legame predeterminato e riconosciuto staticamente (binding statico CORBA)

E se volessimo legami non noti a tempo di sviluppo?

CORBA IDL per MULTILINGUAGGIO

Interface **D**efinition **L**anguage (CORBA IDL) deve **coordinare la identificazione dei servizi richiesti e offerti in diversi linguaggi**

```
interface Factory  //OMG IDL
{         Object create(); // Oggetto CORBA o riferimento
};
```

Questa interfaccia consente di riferire un oggetto di tipo Factory (IDL) e di richiedere su un tale oggetto la operazione **create** (senza *parametri, di in e out*) che restituisce un oggetto CORBA generico (`type Object`, ossia un riferimento all'oggetto `Object`)

Via **IDL** si possono definire **nuove interfacce e nuovi tipi** generali e **non concreti**, secondo necessità, farli riconoscere e registrarli, e poi usarli concretamente nei diversi ambienti di linguaggio

In CORBA non ci sono **creazioni di oggetti** (né Factory): la parte di creazione è negli ambienti di linguaggio e predeterminata (come in C non c'è l'I/O)

CORBA IDL -> STUB E SKELETON

Interface Definition Language (CORBA IDL) permette di generare nei **diversi linguaggi componenti di appoggio** (**stub** e **skeleton**) per la comunicazione ed i dati

Lo **stub** permette di lavorare sul *messaggio dalla parte del cliente* (marshalling) agendo come proxy del cliente

Lo **skeleton** collabora con l'ORB per prendere *la richiesta e adattarla al server* (unmarshalling) trattando richiesta e risposta

DEPLOYMENT

In questo modo lavoriamo producendo un **legame statico** tra ***interfaccia - cliente – servitore*** *(non tra le entità dirette ma tra* ***cliente – servizio*** ***e*** ***servizio - servitore****)*

Gli ***oggetti racchiusi nel loro ambiente*** sono legati alla interfaccia dallo stub e skeleton prima della esecuzione

(stub e skeleton sono oggetti loro stessi? no)

CORBA ADAPTER

Adattatore (**Object Adapter**) **componente di sistema** per superare **disomogeneità** e **differenze** tra le realizzazioni degli **ambienti di servizio** dei diversi servitori detti servant

(NON per la presentazione dei dati)

OA dalla parte del **server**, con compiti tipici:

- funzionalità di ***registrazione*** dell'oggetto
- generazione dei ***riferimenti esterni*** all'oggetto
- ***attivazione*** dell'oggetto *e dei* ***processi interni*** *anche su richiesta*
- ***demultiplexing*** *delle richieste* in modo da disaccoppiarle
- ***invio delle richieste*** (upcall) agli oggetti registrati

I primi adattatori erano Basic (**BOA**), poi Portable (**POA**)

(OA sono oggetti loro stessi? no, come OA sono pseudo-oggetti)

INTERFACE REPOSITORY in CORBA

Interface Repository consente di conoscere tutti i **tipi di dati IDL** e di esplorare le **interfacce** esportate dagli oggetti presenti e disponibili durante la esecuzione

Le interfacce sono traslate nei linguaggi di programmazione diversi (**binding statico**) in cui i componenti sono progettati e compilati (**language mapping**)

IR permette di conoscere e gestire le interfacce presenti **dinamicamente e di decidere al momento della esecuzione (binding dinamico)** su cosa è disponibile

Permette di superare l'approccio statico: ad esempio per *un* ***gateway*** *che permette di accedere alle interfacce CORBA di un ambiente e non può essere ricompilato ad ogni nuova interfaccia*

IR sistema di descrizione dei servizi (non un sistema di nomi)

(IR è un oggetto lui stesso? sì)

ORB e IR in CORBA

In CORBA, **ORB è il tramite per qualunque esecuzione remota e richiesta di operazione tra entità diverse**

Ogni richiesta viene recapitata attraverso l'ORB e poi mediata attraverso l'adattatore dalla parte del servitore

ORB non conosce **informazioni di tipo**, al difuori del suo scope e confinate a stub e skeleton e agli ambienti di linguaggio

Interface Repository svolge la funzione di **catalogo dinamico** delle **interfacce** (non necessario per stub e skeleton **statici**),

ma presente per una **esplorazione dinamica**, in caso durante l'esecuzione si debbano conoscere interfacce non note

Le interfacce devono sempre essere registrate nell'IR al momento dell'uso e sono consultabili

In **caso statico**, non c'è alcun bisogno di IR (fornite dai proxy)

IMPLEMENTATION REPOSITORY (IMR)

Implementation Repository come strumento interno della architettura e poco applicativo per consentire di registrare tutte le implementazioni durevoli dei servant da conservare

Il Middleware mantiene qui traccia di tutti le **implementazioni dei servant** e permette di ritrovarli ed eventualmente rimetterli a disposizione anche in caso di ripartenza

Le interfacce sono presenti nell'IR, le **implementazioni sono tracciate dall'IMR**

IMR permette di conoscere e ritrovare i servant che forniscono determinate interfacce (in modo stabile) e consente di ritrovare in modo preciso gli '**oggetti corrispondenti**'

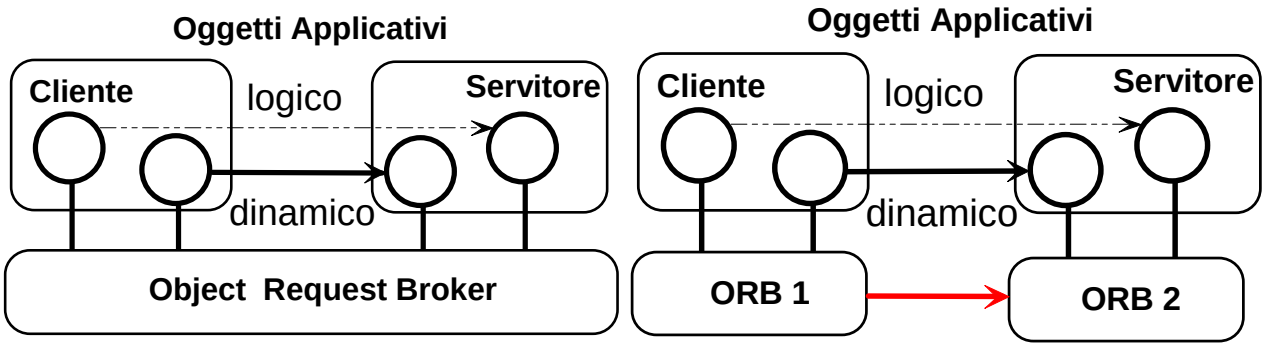
IMR è un **repository interno** per gestire i servant (non un sistema di nomi)

(IMR è un oggetto lui stesso? no)

SISTEMI ORB DIVERSI

ORB per la comunicazione degli oggetti locali (intra-ORB) ed anche per la comunicazione tra ORB diversi inter-ORB)

In un solo sistema CORBA o in più sistemi CORBA coordinando broker diversi tra loro



SISTEMI CORBA DIVERSI

Definizione di standard Inter-ORB per stabilire come integrare e fare interagire senza problemi diversi sistemi CORBA

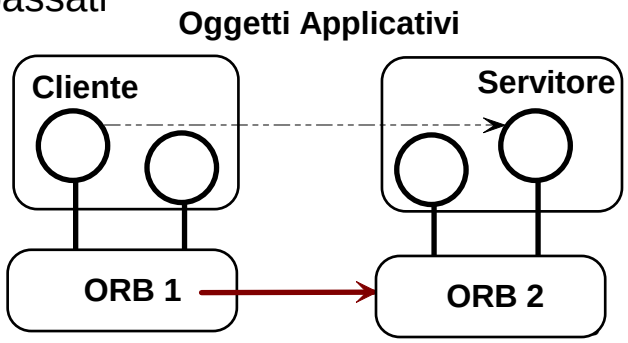
Necessità di protocolli standard per **interoperabilità ORB-to-ORB**

General **I**nter-**O**RB **P**rotocol (**GIOP**) che prevede una standardizzazione del formato dei messaggi

Si specifica quindi il protocollo tra ORB diversi in modo più preciso in termini di architettura e dati passati

Protocollo di comunicazione **binario**: i dati sono ottimizzati e non sono leggibili dall'utente (non sorgente)

Common Data Representation (**CDR**) standardizzato



protocolli GIOP / IIOP

INTER-ORB PROTOCOL: GIOP e IIOP

Definizione (dalla versione 2) di Protocolli InterORB per stabilire come fare interagire diversi sistemi CORBA

protocollo di interoperabilità tra ORB

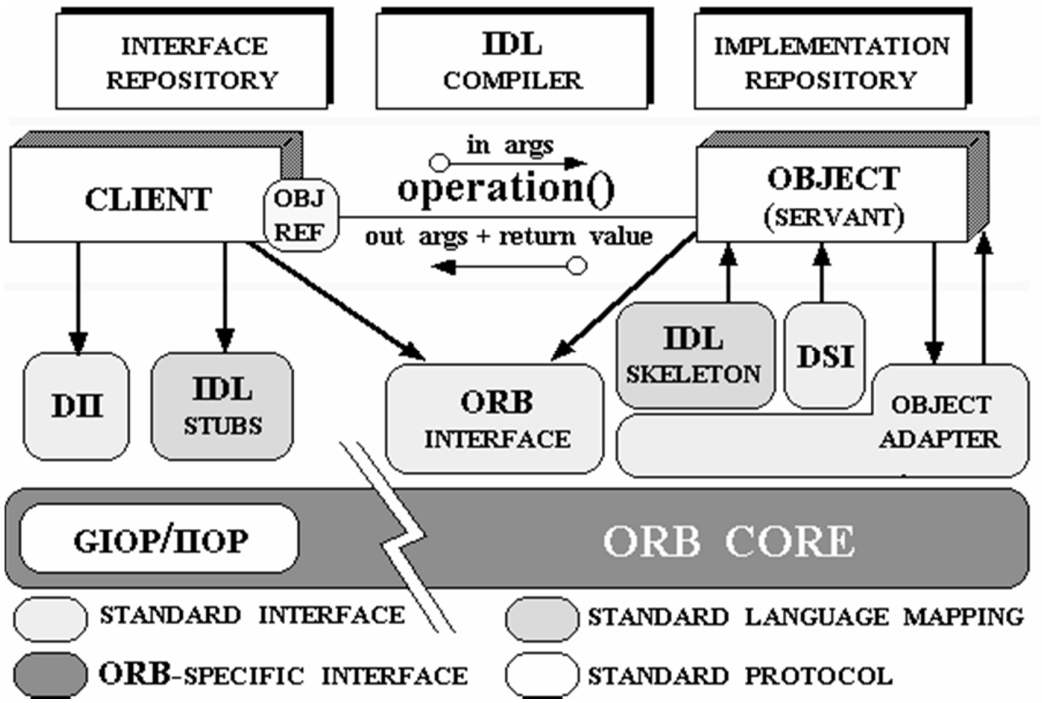
General **I**nter-**O**RB **P**rotocol (**GIOP**) - **Protocollo binario**

Specifica comune della rappresentazione dei dati, del formato dei messaggi, dell'interazione con messaggi di trasporto (assunzioni semantiche: reliable, connection, …)

per Internet su TCP/IP - Internet Inter-ORB Protocol (**IIOP**)

CORBA ARCHITETTURA

Visione di insieme della comunicazione anche tra ORB



CORBA: PSEUDO-OGGETTI

Componenti e pseudo-oggetti di supporto

Stub generato dalla interfaccia nei diversi linguaggi

Skeleton generato dalla interfaccia nei diversi linguaggi

I due componenti realizzano la Static Invocation Interface SII

Ci sono anche altri componenti di architettura, come la interfaccia da cui si generano i proxy, i repository (interface e implementation) per trovare le specifiche dei componenti e delle loro implementazioni, e l'**object reference**

La parte dinamica è incarnata in altri **pseudo-oggetti**

DII, **D**ynamic **I**nvocation **I**nterface, o oggetto Request
generato per la invocazione dinamica al cliente

DSI, **D**ynamic **S**keleton **I**nterface o oggetto ServerRequest
generato per la invocazione dinamica al server

ORB funzioni di base

ORB agisce da coordinatore, da abilitatore e da gestore dei servizi disponibili in un sistema

Le applicazioni portano ad **oggetti** che diventano parte del sistema oltre il **tempo di vita** della **applicazione**

Le **applicazioni** e gli **oggetti** sono stati sviluppati in **ambienti diversi** e rappresentano le **risorse non mobili** che possono richiedere **metodi** e **svolgere operazioni**

ORB interviene in ogni interazione e

- **coordina le richieste degli oggetti clienti di competenza**, in modo **trasparente** dalla posizione e implementazione dell'oggetto remoto
- **facilita** e **gestisce la comunicazione attraverso** l'uso di **riferimenti ad oggetti servant** esistenti
- **supporta** e **controlla** tutta l'**interazione**

ORB funzioni

ORB come facilitatore della interazione ad oggetti in ogni forma, adottando a default interazione sincrona bloccante

ORB per la interazione limita la responsabilità delegando i singoli ambienti di linguaggio per la attuazione finale

CORBA non si occupa di creare e di muovere oggetti: creazione esplicita esterna di oggetti di servizio (servant)

ma di **riferirli da e tra gli ambienti implementativi** attraverso **riferimenti remoti CORBA** ottenuti attraverso:

- conversione di **riferimenti in stringhe** e viceversa (oggetti riferiti e traslati in stringhe - stringhificazione, e viceversa)
- utilizzo di **directory di oggetti,** con uso di servizi di nome (Trading e Naming service)
- **passaggio di parametri** di **riferimenti** a **servant**

CORBA IDL

INTERFACE DEFINITION LANGUAGE (**OMG IDL**) introdotti per garantire flessibilità su piattaforme eterogenee

Sono **linguaggi dichiarativi** per **specificare interfacce** e dati

Molti IDL diffusi sono **procedurali**

* OSI **ASN.1** / GMDO
* ONC **XDR** (SUN RPC)
* **M**icrosoft **IDL**

CORBA IDL è un linguaggio **object-oriented** (***derivato dal C++***)

Ovviamente i diversi IDL sono **non compatibili** tra di loro, anche se spesso sono diversi solo per questioni di **sintassi** e di sistemi di **identificazione e nomi delle entità**

CORBA IDL

CORBA IDL come puro **linguaggio di descrizione dei dati e delle interfacce dei metodi**

- descrizione delle sole **definizioni dei metodi**
- **interfacce** come insiemi di metodi ed attributi
- **ereditarietà multipla** delle interfacce
- definizione **eccezioni**
- gestione automatica degli **attributi**
- **mappaggi** per **linguaggi diversi** ed ambienti diversi

Il compilatore può ottenere automaticamente stub per clienti/servitori anche usando linguaggi diversi

Bisogna considerare i diversi **linguaggi di mapping** **per i riferimenti agli oggetti server**

ESEMPIO di CORBA IDL

```
module Stock
{exception Invalid_Stock {}; exception Invalid_Index {};
 const  length = 100;

 interface Quoter {
   attribute float quote; readonly attribute float quotation;
  long get_quote(in string stock_name) raises (Invalid_Stock);
 };
  interface SpecialQuoter: Quoter {
   attribute float quotehistory [length];
   readonly int index [length];
   long get_next (in string stock_name) raises (Invalid_Index);
   long get_first(in string stock_name) raises (Invalid_Index);
 };
 interface CancelQuoter: SpecialQuoter {
   long cancelhistory (out float cancelledquote [length])
 };
}
```

SUPPORTO a CORBA IDL

Per ogni attributo si mettono a disposizione automaticamente funzioni di accesso adatte alle operazioni consentite (_get per letture e _set per scritture)

```
attribute float quote;
float   _get_quote ();
void    _set_quote (in float q);
readonly attribute ind index;
float   _get_index ();
```

Per ogni eccezione, lo stato (completion_status) fornisce informazioni semantiche sul completamento

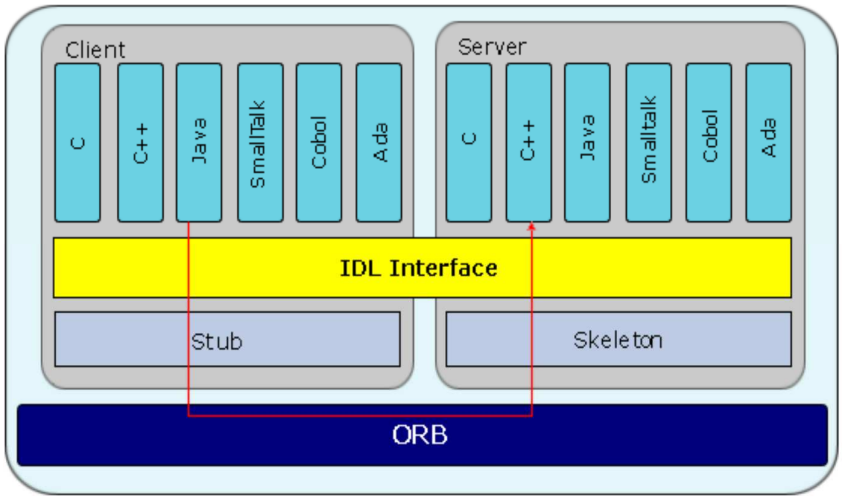
```
COMPLETED_YES,
COMPLETED_NO,
COMPLETED_MAYBE
```


CORBA IDL

Linguaggio per definire le interfacce in CORBA, indipendente dai **singoli linguaggi concreti** di programmazione

Naturalmente si deve **passare da CORBA** ai **linguaggi concreti specifici** (**language mapping**)

CORBA lascia spazio agli **ambienti di mapping.**
Per la **creazione di servant**, la responsabilità è dei diversi language mapping



AMBIENTE CORBA IDL

CORBA è un **ambiente** in cui **usiamo riferimenti remoti e non muoviamo oggetti (oggetti fissi)** vista ***la eterogeneità dei singoli ambienti concreti di deployment***

I **riferimenti remoti** permettono di richiedere operazioni ad altri componenti con interfaccia CORBA nota

Ogni oggetto ha **una interfaccia (granularità grossa)**

Le **interfacce** prevedono: **attributi**, **metodi**, **eccezioni**

(*attributi acceduti attraverso operazioni di **get** e di **set***)

(*operazioni con argomenti di **in** o/e **out***)

Le interfacce in **ereditarietà multipla**

Le **interfacce** sono anche racchiuse in **moduli**
(per aggregazioni logiche)

ALTRO ESEMPIO di CORBA IDL

```
module ContoCorrente {
struct movimento { string data; float importo;};
exception RossoException {string message;};
typedef sequence <movimento> lista_op;
interface Conto {
  float saldo (in string cc);
  lista_op estratto (in string cc);
  void prelievo (in string cc, in float importo,
     out float saldo) raises RossoException;
  Conto contogemello(); // restituisce un oggetto };
};
```

Parametri passati per valore (oggetti **CORBA per riferimento**)

Problema in gestione parametri **`out`** e **`in out`**

DATI in CORBA IDL

Tipi in CORBA

Object Reference (riferimenti ad **oggetti o interfacce**)
vs. anche in ereditarietà tra oggetti CORBA

Value (copia di valori) ed **Exceptions**

Basic values short, long, ushort, ulong, float, double, char, string, boolean, octet, enum, Any

Constructed values Struct, Sequence, Union, Array

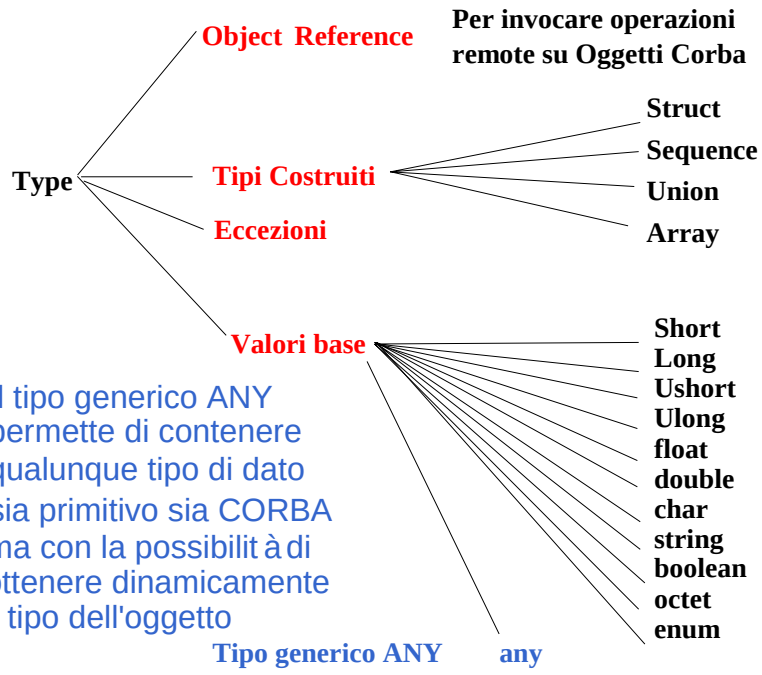
Any come tipo generale che contiene qualunque tipo, primitivo o da interfaccia CORBA (esaminabile durante la esecuzione)

Object by value *(CORBA 3)*

*Oggetti che **non** possono essere acceduti da remoto ma vengono passati solo **per copia** da un ambiente ad un altro superando le eterogeneità dei diversi ambienti (nessun riferimento remoto ad essi)*

TIPI in CORBA IDL

Tipi in CORBA IDL



Il tipo generico ANY permette di contenere qualunque tipo di dato sia primitivo sia CORBA ma con la possibilità di ottenere dinamicamente il tipo dell'oggetto

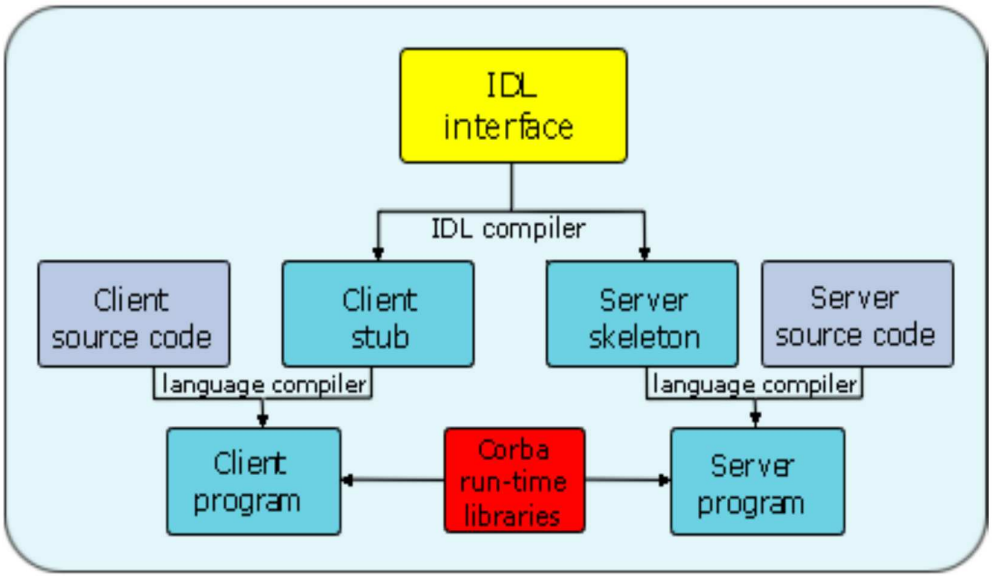
I tipi di CORBA IDL sono poi traslati nei tipi dei diversi linguaggi di programmazione ottenuti per i diversi language mapping

Il tipo Object (IDL) rappresenta qualunque tipo di oggetto CORBA senza alcuna indicazione del tipo specifico

Da CORBA IDL ai Linguaggi

Si usano **strumenti** per derivare da CORBA IDL i diversi componenti necessari per il progetto e la esecuzione nei **diversi linguaggi di mapping**

stub e **skeleton** + **file helper** e **vari di aiuto** (**holder**) + altre operazioni



CORBA Language mapping

CORBA definisce

interfacce (in ereditarietà), **eccezioni**, **metodi** con **parametri come oggetti** di tipi definiti e con **modi** diversi (**in**, **out**, **in out**)

I diversi linguaggi devono o aggiungere **concetti**, o armonizzare le **proprie strutture** per ottenere la **conformità alla interfaccia** e garantire la **operatività run-time (se sono OO anche ereditarietà da integrare)**

Necessità di avere consistenza con i tipi concreti del linguaggio e di potere trattare il modello CORBA

funzioni di trasformazioni varie fornite automaticamente per trattare i tipi, per mettere insieme le strutture in modo semplice,

Oltre a molte altre funzioni di supporto (naming, trading, e metodologie suggerite per lo sviluppo) usabili dall'utente

CORBA vs LINGUAGGI: HOLDER

Uso di **holder** in JAVA come linguaggio, in cui non ci sono parametri di output)

ad esempio

```
public  final Class SaldoHolder …
{public float value;
public SaldoHolder() {}
float _read() {return value;}
void _write(float valore) {this.value = valore;}
};
```

per i parametri di **out** e **in out** (anche altri aiuti: helper)

In genere, ogni **linguaggio** concreto deve creare tutto quello che è necessario per favorire lo sviluppo nel suo ambiente

CORBA HELPER

Uso di **helper** per Language mapping: in Java funzioni per

- **armonizzare e trattare i tipi del linguaggio e i tipi di CORBA**

*In Java il tipo **CORBA Object** è mappato in org.omg.CORBA.Object*

funzioni di **narrow**-ing che portano dal tipo CORBA Object al tipo specifico atteso e definito nella interfaccia

funzioni per trattare trasformazioni dal tipo di CORBA per il tipo di interesse specifico

- **fornire funzioni di utilità varia**

funzioni per **leggere da e scrivere** su stream oggetti di un tipo (associato alla interfaccia CORBA), per **trattare il tipo dinamicamente** durante l'esecuzione, …

Ogni **linguaggio** deve garantire interoperabilità CORBA

DISPONIBILITÀ di AMBIENTI CORBA

Molto ampia e ancora in crescita

Object Broker	DEC
ORB	HP
DSOM	IBM
Orbix	IONA
Visibroker	Borland
(DOM Facility)DOE	Sun Studio Sun
PowerBroker	ExperSoft
JacORB, ...	**Strumenti open source**

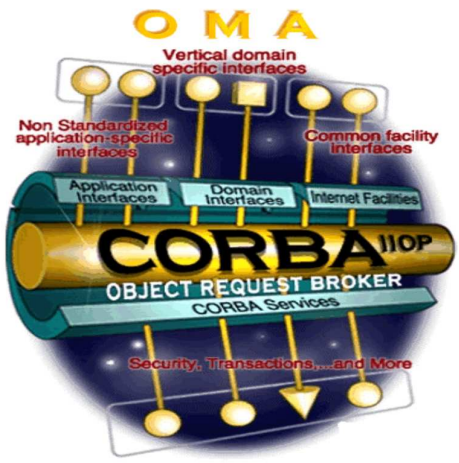
Anche se la curva di apprendimento è elevata e ci sono overhead nelle prestazioni

Middleware

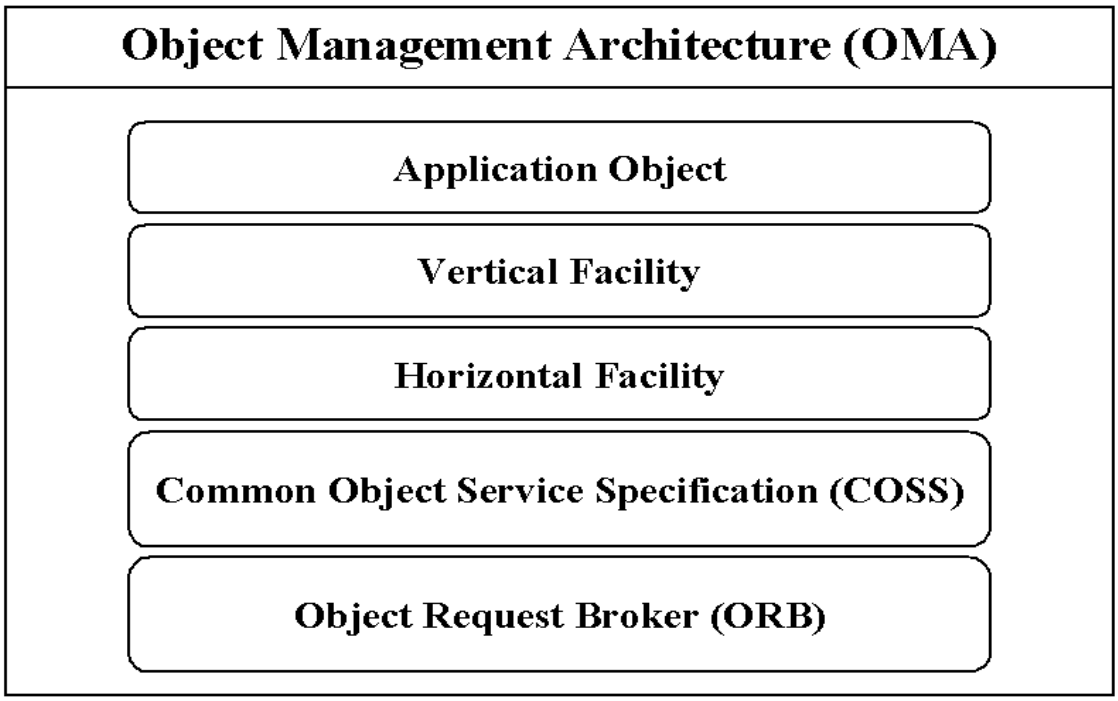
I middleware più usati devono prevedere risposte anche ad esigenze spicciole e a dettagli ulteriori

Gli utilizzatori di CORBA si aspettano di potere:

- progettare **velocemente nuovi componenti**
- **integrare** vecchi componenti **legacy**
- potere utilizzare **strumenti esistenti** e componenti di **ausilio pronti**
- potere **integrare** le **applicazioni** con nuove **facility disponibili**
- avere un middleware **capace di ospitare servizi senza interruzione (QoS) e con tempo di vita senza limiti**



ARCHITETTURA CORBA



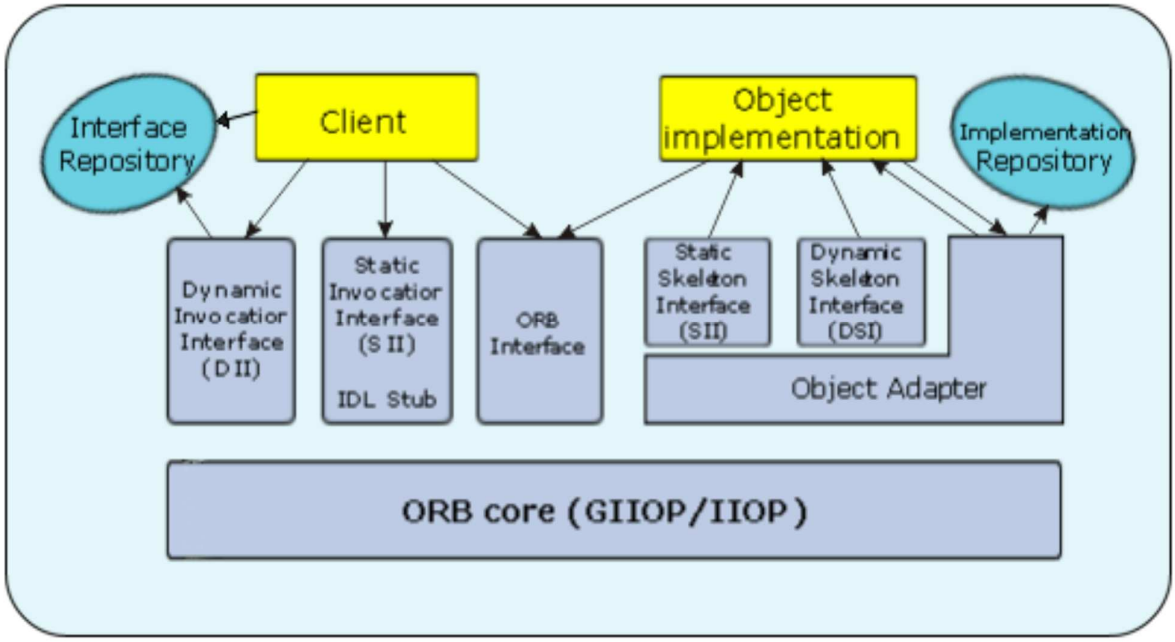
COMPONENTI DI CORBA

I componenti essenziali di CORBA

* **Object Request Broker** (**ORB**)
* **Interface Definition Language** (**IDL**)
* **Basic (e Portable) Object Adapter** (**POA**)
* **Static Invocation Interface** (**SII**)
* Dynamic Invocation Interface (DII)
* Interface e Impl. Repository (IR e IMR)
* Protocolli per Integrazione (GIOP)

CORBA ARCHITETTURA

Visione di insieme della architettura base di supporto al servizio



PROGETTO PROGRAMMI CORBA

Si devono specificare le **interfacce comuni** a server e clienti

Dopo avere generato gli **stub** e **skeleton**

Si implementano le **classi server**

Il server **deve registrarsi**

Si implementano le **classi client**

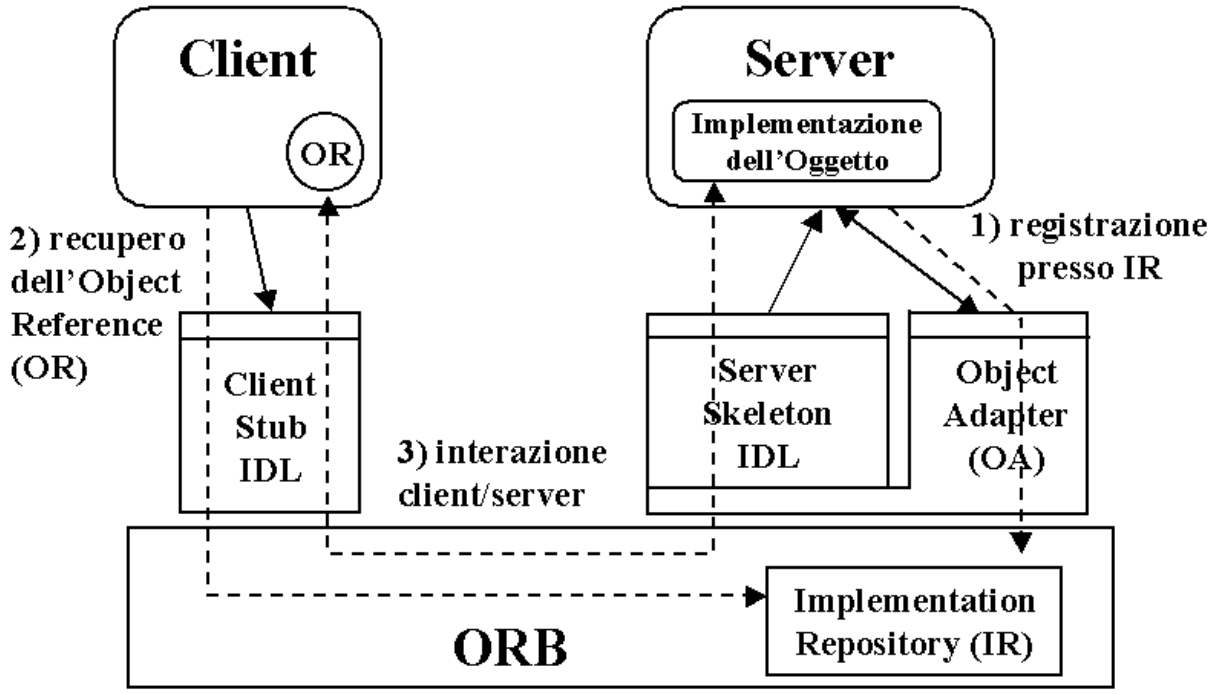
Si va alla **esecuzione**

Necessità di **riferimenti remoti** del cliente per ritrovare

il **server**, i **componenti**, … , e in generale
la intera **infrastruttura di supporto**

Deployment: quanti ORB? Dove sono? Come si raggiungono?
O su **ogni nodo** (API locali), o **ORB** centralizzato, o **più** server
Con che QoS? Che tolleranza ai guasti?

COMPONENTI DI CORBA



ORB INTERFACE in CORBA

ORB si può intendere come l'insieme di classi che permettono un **buon supporto ad oggetti remoti**

Funzioni di conversioni varie

funzioni per trasformare gli **ObjectReference (o Object Interface)** in stringhe (per mantenerli in modo comodo) e viceversa

```
Interface ORB {
    string object_to_string (in Object obj);
    Object string_to_object (in string str); }
```

Con la stringhificazione, possiamo passare da una forma all'altra anche tra ambienti diversi

Anche funzioni per inizializzare i vari OA, per ritrovare servizi necessari, funzioni di base, ecc. ecc.

ORB INTERFACE in CORBA

ORB Funzioni varie

`ORB Initialize` per il boot

```
ORB ORB_init(inout arg_list arguments,
    in ORBid ORB_identifier)
```

per il collegamento iniziale all'ORB da parte dei clienti

Anche una serie di funzioni per ritrovare il **contesto di default** e ottenere i riferimenti per i **servizi di base** (IR, naming service, …)

```
typedef string ObjectID;
typedef sequence <ObjectID> ObjectIDList;
ObjectIDList list_initial_services ();
Object resolve_initial_references (in string ObjectID);
```


OGGETTI INIZIALI NOTI in CORBA

La funzione per ottenere oggetti di base (ObjectReference) permette in genere di accedere ad esempio:

```
Object resolve_initial_references (in string ObjectID);
```

oggetti CORBA di supporto ritrovati via Servizi Iniziali

"**RootPoa**", "**POACurrent**", "**InterfaceRepository**",

servizi CORBA di supporto

"**NameService**", "TradeService", "NotificationService", …

politiche CORBA correnti

"ORBPolicyManager", "TransactionCurrent", "PolicyCurrent" …

Un object reference in stringa potrebbe essere:

IOR:000000000000001949444c3a696f722f53696d706c654f626a656374
3a312e3000000000000000010000000000000030000100000000000a737
465656c7261696e00079e00000018afabcafe000000023bd4cf8d0000000800000000000000000

Object Reference in CORBA

Gli **Object Reference** permettono di riferire **una istanza di un servizio remoto** (**uno stub**): sono **opachi** e **non** sono manipolabili dai clienti ma solo dall'ORB
(*eventualmente integrati con la gestione della persistenza*)

Gli **Object Reference** sono riferimenti ad istanze di **Object** di CORBA

Operazioni fornite dalla ***Object Interface*** sono molte per consentire di lavorare in modo trasparente e visibile

```
get_implementation, get_interface,
is_nil, non_existent, is_a, is_equivalent,
hash, duplicate, release,
create_request, get_domain_manager,
get_policy, set_policy_overrides, …
narrow, this, …
```


Object Reference in CORBA

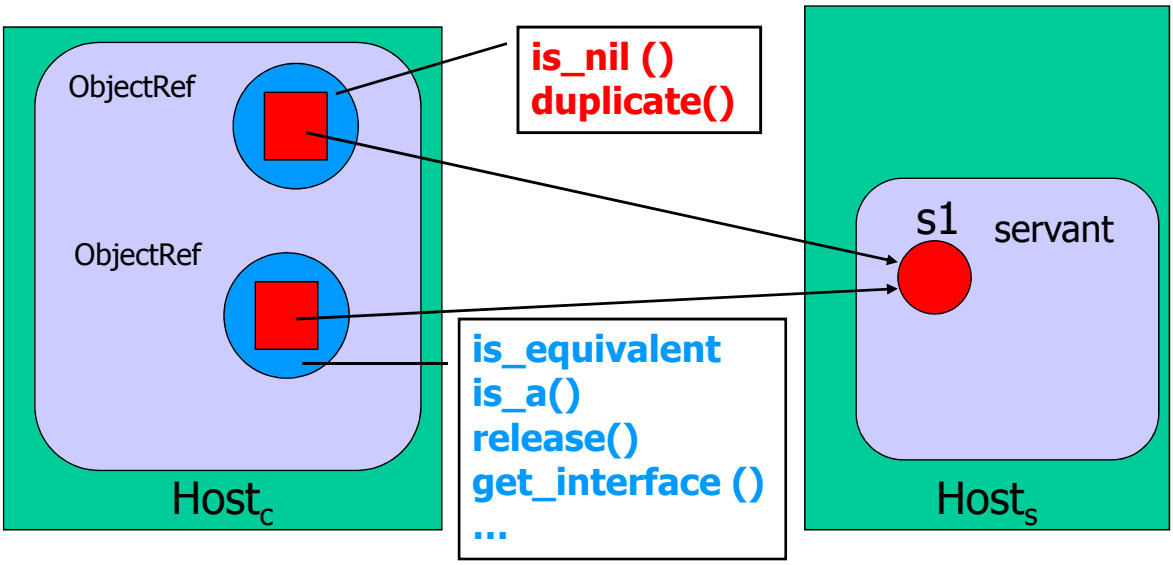
Gli **Object Reference** di CORBA ereditano da

CORBA::Object interface

```
interface Object {
// operazioni per gestione degli oggetti
Object duplicate (); void release ();
// operazioni per conoscenza oggetto
Object get_implementation ();  Object get_interface ();
// operazione di esistenza e riferimento
boolean is_nil ();
boolean non_existent ();
boolean is_equivalent (in Object other_object); // stesso obj?
boolean is_a (in string repository_id); //implementa?
Object create_request (in Object); // crea oggetto request
// ...
}
```

Object Reference in CORBA

Gli **Object Reference** sono opachi e si deve anche consentire **una corretta operatività e le funzioni di gestione necessarie**



VERSIONI di CORBA

CORBA mantiene i **componenti essenziali** anche nella sua evoluzione ma si arricchisce di ***strumenti*** e di ***componenti*** per *ovviare ai* ***problemi man mano delineati*** e per *fornire un* ***migliore supporto***

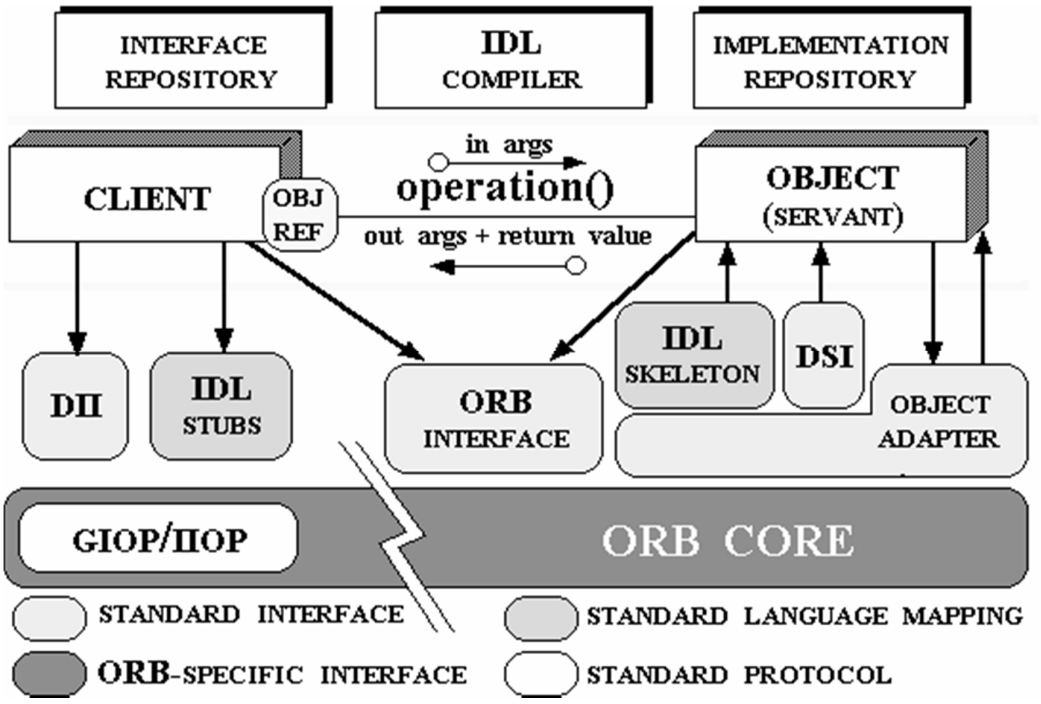
Componenti essenziali sempre gli stessi

- Interazione tra ambienti di linguaggi diversi
- Aiuti per l'uso di ambienti di linguaggi diversi
- Strumenti per ottenere **QoS** in ambienti di linguaggi diversi
- Nuove utilità generali e per domini specifici
- Nuove realizzazioni ed integrazione con diversi ambienti di sviluppo esistenti

CORBA 3 (2000 - …)

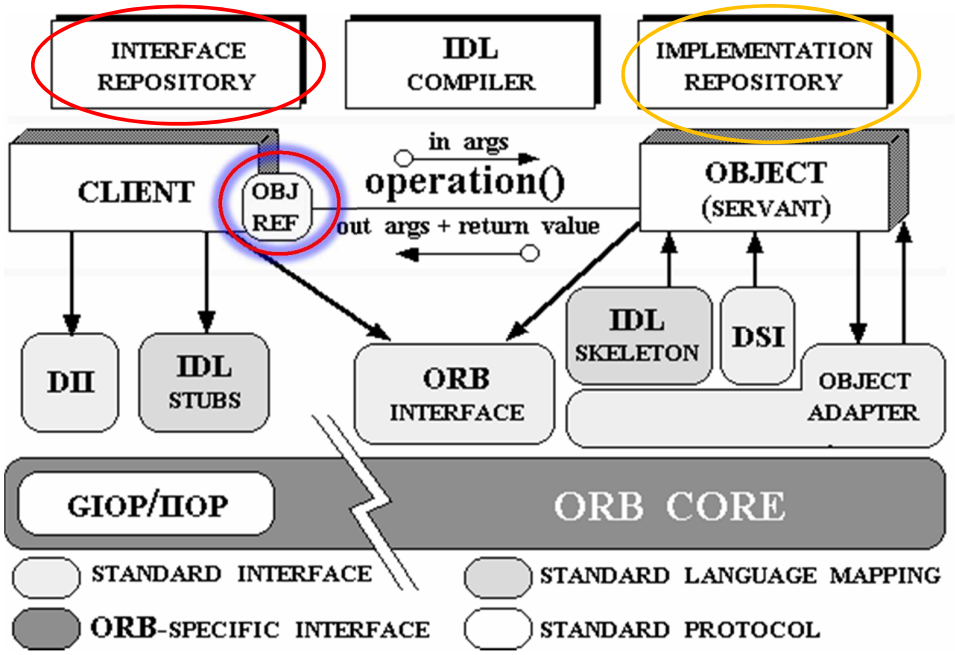
CORBA ARCHITETTURA: dettagli

Visione di insieme della implementazione della architettura



CORBA REPOSITORY

IR deve registrare le interfacce di tutti i servizi presenti, **IMR** i servant, gli **Object reference** permettono di ottenere i servizi



BINDING STATICO in CORBA

Static Invocation Interface (SII)

*Il compilatore e gli strumenti risolvono le chiamate prima della esecuzione con la creazione di **stub** e **skeleton***

Tutte le invocazioni sono controllate in anticipo e sicure

nessun controllo dinamico nei confronti della interfaccia visto che i proxy lo prevedono in modo statico

Il **cliente** si *lega allo stub* e fa la richiesta tramite riferimento dopo essersi collegato all'ORB (**invocazione sincrona**)

Il servitore (detto **servant**) si è *legato allo skeleton* e viene attivato dall'object adaptor (POA) per le richieste

Non c'è alcun legame tra **client** e **servant**: a richieste successive anche servant diversi, ma stessa interfaccia

In caso il (un) **servitore non sia attivo, il POA lo attiva e gli passa la richiesta**

SEMANTICA in CORBA

La normale modalità è la **sincrona bloccante**

In caso di guasti o problemi, il cliente riceve una eccezione di quelle previste dalla interfaccia

Semantica at-most-once

In CORBA la **invocazione sincrona** si basa su proxy statici usati come intermediari

Ovviamente questo può essere molto limitante

La invocazione **sincrona** statica ha un costo 'molto' limitato (se si prevedono operazioni a grana grossa)

Sono necessarie altre modalità?

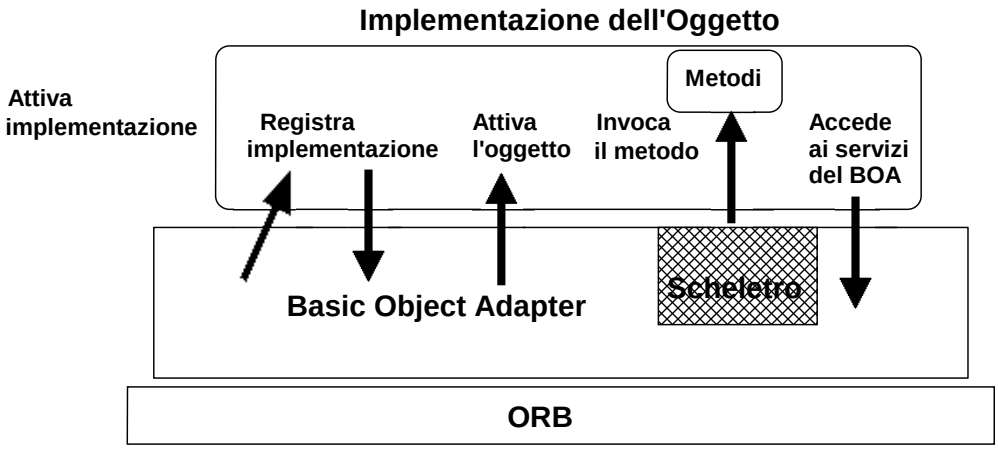
Oneway in IDL: *nessuna risposta (best effort)* *deprecata*

COMPONENTI CORBA: ADATTATORE

Gli **Adattatori** sono i componenti responsabili della flessibilità di CORBA

I **molteplici adattatori** devono arrivare alla implementazione dei diversi oggetti servitori e comandare la esecuzione concreta

Si parla di **servant** per la **parte passiva** che esprime negli oggetti concreti le funzioni del server



Funzioni di ADATTATORE

Gli **Adattatori** controllano la **esecuzione delle operazioni** nei server attraverso il concetto di **servant**

USO di SERVANT

Un **servant** è la **parte di oggetto che mette a disposizione il codice da eseguire su richiesta di un cliente** (entità fortemente dipendenti dal linguaggio di programmazione e dalla specifica implementazione del servitore)

La reale implementazione del servizio in un linguaggio

Il POA ha il compito di comporre la immagine dell'oggetto CORBA server

Un POA (su un nodo) potrebbe controllare:

- **un unico servant**
- **anche molti diversi servant da fornire alle diverse richieste**

Un **POA** decide i **propri servant** e la **politica di gestione**

COMPONENTI CORBA: ADATTATORE

Gli Adattatori controllano la esecuzione **del server astratto** attraverso i **servant concreti che lavorano sul codice del servizio**

MOLTI MODI DI ATTIVAZIONE NEL SERVER

attivazione per ogni richiesta **(thread_per_request)**

un processo viene creato nell'oggetto per servizio

attivazione iniziale di un pool **(pool di thread)**

ogni oggetto riceve il suo processo da un pool creato inizialmente senza il costo della creazione

attivazione per sessione **(thread_per_session)**

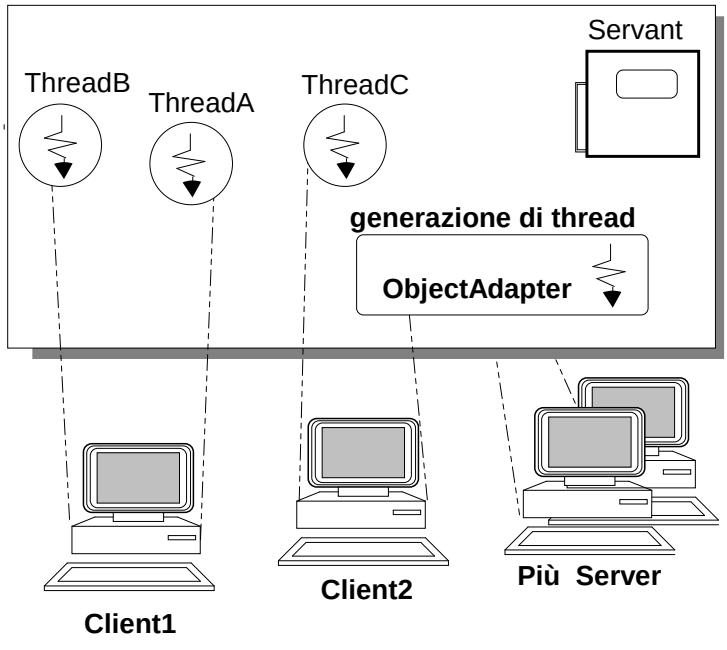
ogni cliente ha un processo dedicato alla interazione

Anche altre modalità: ***un thread per ogni servant*** **(thread_per_servant)**

una sola attivazione **per più oggetti server**

(shared server) contemporaneamente

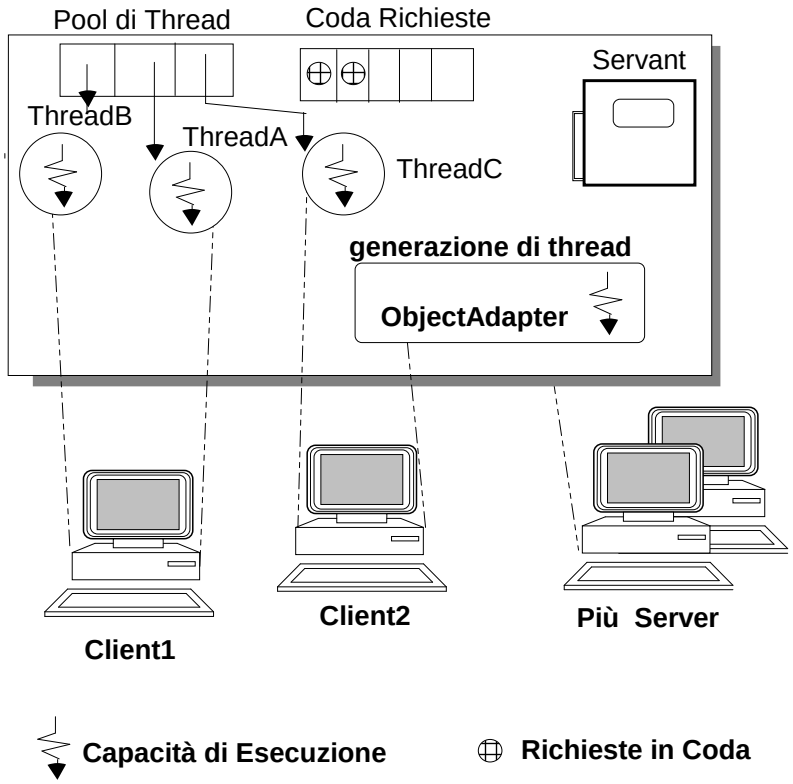
Attivazione THREAD-PER-REQUEST



Ogni richiesta riceve un thread attivato by-need

Costo elevato della attivazione che pesa su ogni operazione

Attivazione THREAD-POOL

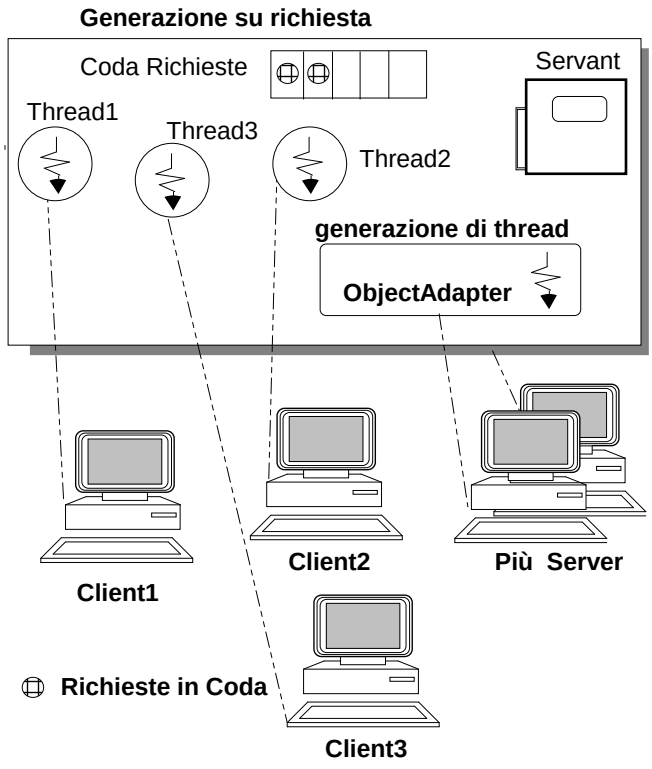


Ogni richiesta riceve un thread da un pool di processi pre-creati

In caso non ce ne siano, si aspetta il primo libero

Costo inferiore, ma anche elevata attesa in caso di traffico non previsto

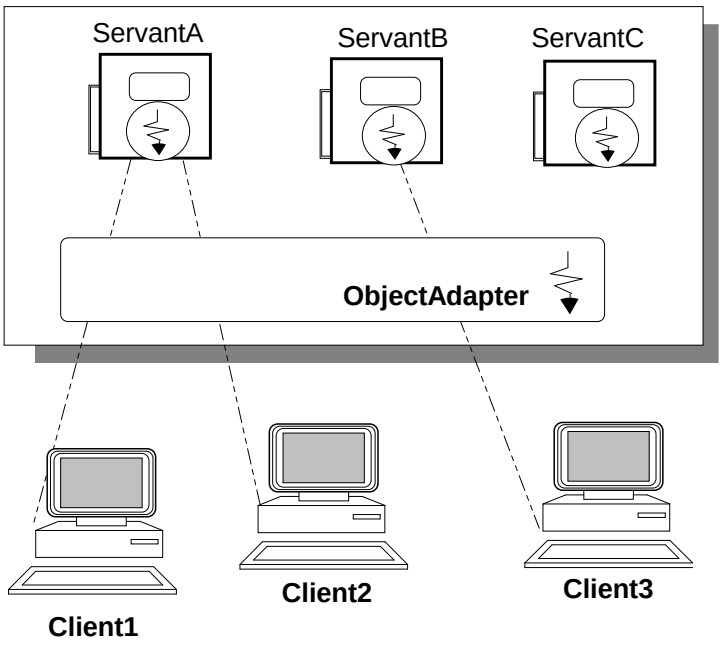
Attivazione THREAD-PER-SESSIONE



Ogni cliente riceve un thread attivato all'inizio della sessione di lavoro

Si limita il parallelismo del servizio

Attivazione THREAD-PER-SERVANT



Ogni oggetto viene incarnato in un servant che prevede un thread dalla prima attivazione e risponde solo attraverso quello

Si limita il parallelismo in base al numero dei servant

MIDDLEWARE per CONTINUITÀ

CORBA è **middleware** per favorire ed abilitare un tempo di vita infinito alle ricorse della organizzazione e cerca di supportare questo punto di vista

Middleware per continuità di servizio

L'infrastruttura ha evidenziato un **contratto basato sulla interfaccia** e può ottimizzare le risorse di implementazione a secondo di politiche che favoriscono indicatori specifici

Il middleware può **bilanciare il carico dei vari servant per un servizio ottenendo migliore throughput**

Il middleware può **in caso di guasto di alcune sue parti, o di altre strategie (e.g., di localizzazione, bilanciamento di carico) per dirigere le richieste verso altre risorse (no downtime)**

Il middleware può **dirigere il carico verso servant in altri sistemi CORBA per un servizio differenziato**

COMPONENTI DI CORBA

I componenti essenziali di CORBA

* Object Request Broker (ORB)
* Interface Definition Language (IDL)
* Basic (e Portable) Object Adapter (POA)
* Static Invocation Interface (SII)
* **Dynamic Invocation Interface** (**DII**)
* **Interface e Impl. Repository** (**IR e IMR**)
* **Protocolli per Integrazione** (**GIOP**)

BINDING DINAMICO in CORBA

Dynamic Invocation Interface (DII) e
Dynamic Skeleton Interface (DSI)

necessari per operare senza avere previsto **staticamente il legame** con l'**interfaccia**, cioè per legarsi *ad* ***interfacce che non esistono al momento della compilazione (ma successivamente)***

comportamento DINAMICO

In generale, il **comportamento dinamico** permette ad una applicazione di adeguarsi a **situazioni non previste** durante lo sviluppo, o meglio a interfacce non note allo sviluppo (e allungare **il tempo di vita della applicazione** stessa)

In questo caso, il cliente e il servitore possono agganciarsi ad interfacce non previste al momento del lancio della applicazione stessa

BINDING DINAMICO in CORBA

Dynamic Invocation Interface (DII) e
Dynamic Skeleton Interface (DSI)

comportamento DINAMICO

Il cliente ed il servitore, che non hanno previsto proxy, si appoggiano a run-time su degli **pseudo-oggetti necessari** per la **operatività (con necessità di controlli run-time sulla correttezza dei tipaggi)**

Le interfacce a cui si fa riferimento nel caso dinamico devono essere registrate e disponibili al momento dell'uso dinamico

Interface repository per la consultazione ed eventuale conoscenza dell'interfaccia di interesse (che deve essere presente al momento dell'uso)

CLIENTE DINAMICO in CORBA

comportamento DINAMICO CLIENTE - DII

- Il cliente riceve un **`ObjectReference`** dinamico, ossia per cui non ha previsto proxy
- crea un oggetto **`Request`** dopo avere recuperato la sua Interfaccia
- usa l'oggetto **`Request`** come intermediario per la interazione con i servant che la implementano e per chiedere metodi a questi

La **`Request`** può essere usate per **molte richieste dinamiche** per la **stessa interfaccia**

In caso cliente si possono assumere invocazioni meno sincronizzate e più varie tra il cliente e il servitore (forme diverse di *asincronicità*)

BINDING DINAMICO in CORBA (DII)

ORB permette di creare, e gestire una **richiesta dinamica** e le **invocazioni** attraverso uno **pseudo-oggetto Request**

```
pseudo typedef long ORBstatus;
ORBstatus create_request {// Pseudo IDL
  in Object         obj // oggetto della operazione
  in Context        ctx // contesto della operazione
  in Identifier     operation // nome operazione sull'oggetto
  in NVList         arg_list // argomenti della operazione
  inout NamedValue  res // risultato della operazione
  out Request       req // richiesta creata per la operazione
  in Flags          req_flags // flag per la operazione
}
```

Si può sempre preparare una richiesta **Request** orientata verso una operazione per arrivare ad una invocazione

La richiesta è disponibile e il cliente può usarla a secondo di quello che vuole ottenere come metodi

DII: REQUEST per BINDING DINAMICO

API per comporre la richiesta mediante uno **pseudo oggetto Request che incarna la DII**

```
pseudo interface Request {// Pseudo IDL
  Status add_arg (in Identifier name, in TypeCode arg_type,
                  in void *value, in long len, in Flag arg_flag );
  Status invoke (in Flags invoke_flags // flag invocazione);
  Status get_result (in Flags flags // flag estrazione risultato);
  Status send_oneway (in Flags flags  // flag invocazione); …
  Status send_deferred (in Flags flags  // flag invocazione);
  boolean poll_response (in Flags flags  // flag invocazione);
  Status get_response (in Flags response_flags //flag risposta);
}
```

La invoke permette la invocazione dalla richiesta **Request,** allo sblocco di va ad acquisire il risultato (poll verifica arrivo risposta, get sincrona estrae)

La richiesta viene preparata e la **chiamata dinamica** comporta un costo più elevato rispetto alla statica sincrona …

PSEUDO OGGETTI in CORBA

Nella architettura CORBA ci sono alcune entità di supporto che sono detti **pseudo-oggetti: Request** è un esempio

Gli pseudo-oggetti sono entità necessarie all'utente per potere ottenere la operatività senza diventare oggetti CORBA:

- *non hanno un riferimento CORBA (non sono oggetti)*
- *sono confinati all'interno dell'ORB specifico*
- *non si producono helper, holder, ecc. nei language mapping*

Gli pseudo-oggetti sono dei facilitatori con descrizione CORBA di alcune entità di sistema (non specificate dalla applicazione) e che una applicazione può usare per i suoi scopi

Possono anche essere mappati diversamente nei diversi linguaggi e presenti solo in alcuni ambienti di linguaggio

BINDING DINAMICO in CORBA (DII)

Un utente **deve operare** attraverso un **oggetto Request sottoponendolo all'ORB** per la esecuzione delle operazioni:

- creazione della *richiesta*
- impostazione/controllo dei ***parametri* in** (nome, tipo, valore)
- impostazione della *risposta* (tipo)
- impostazione delle eventuali *eccezioni*
- impostazione degli eventuali *contesti*
- **invocazione vera e propria** (anche **oneway** o **deferred**)
- *verifica* di eventuali *eccezioni* (dopo il completamento)
- *estrazione* di tutte le *informazioni* dalla *richiesta*:

  parametri di ***out***, ***in out***, ***valore di ritorno***

Il riferimento remoto permette di trovare ed esplorare **l'interfaccia** attraverso il **repository IR**

SEMANTICA INVOCAZIONI in CORBA

La normale modalità è la **sincrona bloccante**

In caso di guasti o problemi, il cliente riceve una eccezione di quelle previste dalla interfaccia

Semantica at-most-once

La invocazione **sincrona** statica introduce meno passi della **dinamica** corrispondente

Nuove modalità previste per la sola **invocazione dinamica**:

Oneway Invocation: nessuna risposta (semantica best effort)

Deferred Synchronous: risposta da ritrovare in tempi successivi con attesa solo quando necessario (at-most-once)

uso di get e poll per la risposta

Si possono mescolare le modalità statiche e dinamiche?

SEMANTICA INVOCAZIONI DINAMICHE

Diverse modalità di azione sullo pseudo-oggetto Request

Sincrona bloccante

`invoke()  … get_result()`

Oneway Invocation

`send_oneway()`

Deferred Synchronous

`send_deferred()…`

```
/* molte operazioni */      poll_response() …
/* si ottiene risultato */  get_response ();
```

Nel caso di invocazione dinamica, tutte le garanzie che derivano da un controllo statico (stub) non sono state ottenute **e sono a carico del cliente alla invocazione che è tenuto a fare le necessarie verifiche: di corretto tipo dei parametri, di eccezione, ecc.**

SERVER DINAMICO in CORBA

comportamento DINAMICO SERVER - DSI

- Il servitore decide di volere implementare una **nuova interfaccia** per cui non ha lo stub generato staticamente
- Usa uno **pseudo-oggetto `ServerRequest` dinamico** come intermediario con due compiti fondamentali
  - **Registrazione** della propria intenzione di **servizio** al POA e ORB (in modo preliminare a qualunque invocazione)
  - Intermediario per la **invocazione delle singole richieste** di servizio (che vengono svolte da una funzione generale di nome **`invoke`** registrata al POA). La **`invoke`** deve controllare i parametri e la correttezza dinamicamente

La **`ServerRequest`** può essere usato come abilitatore per tutti **i metodi della interfaccia**

BINDING DINAMICO in CORBA (DSI)

Un server che voglia fornire una **implementazione dinamica** di **operazioni** deve definire un **comportamento DINAMICO** via **Dinamic Skeleton Interface (DSI)**

Il servant si appoggia a run-time sul POA per registrarsi come una implementazione possibile e valida, associata all'interfaccia di interesse

Si usa uno **pseudo-oggetto ServerRequest** che **registra** la realizzazione al POA e **permette al POA di considerarla** come una di quelle ammissibili e gestibili

L'operazione dinamica richiede un **maggiore controllo di correttezza** (se possibile) dei parametri che non sono stati verificati staticamente dal supporto di linguaggio

Ogni invocazione, statica o dinamica, può essere diretta al nuovo servant *così registrato* che usa lo **pseudo-oggetto** come tramite per **parametri e risultato**

BINDING DINAMICO in CORBA (DSI)

Per potere fornire la operazione, il server deve usare una **offerta ServerRequest dinamica di operazione**

```
pseudo interface ServerRequest {// Pseudo IDL
 readonly attribute Identifier operation_name;
 readonly attribute OperationDef operation_definition;
 void parameters(inout NVList params);
 Context ctx();
 void set_result(in Any val);
 void set_exception(in Any val);
};
```

Lo pseudo-oggetto **`ServerRequest`** deve essere registrato al **POA** (dall'ORB a cui viene presentato) per rendere noto che esiste una nuova implementazione della specifica operazione di una interfaccia (da parte dell'oggetto stesso)

ORB e POA giocano un ruolo fondamentale in DSI

BINDING DINAMICO in CORBA (DSI)

L'oggetto server deve implementare una **`invoke`** da parte del POA (per eseguire i metodi) in forma di una **`callback`**

l'ORB richiede al POA di usare questa **`invoke`** per ottenere la esecuzione generica da parte del servant; ORB e POA passano la richiesta all'oggetto, che ha la responsabilità di eseguire il metodo realizzato (appoggiandosi sul contenuto della **`ServerRequest`**)

Ovviamente in questo caso non sono stati fatti i controlli del caso statico dallo skeleton

il metodo di **`invoke`** deve andare ad **acquisire il nome del metodo**, i **parametri**, **controllarli**, eseguire la **logica**, e produrre i **risultati** (da controllare in tipo)

In caso di problemi deve passare le necessarie **eccezioni**

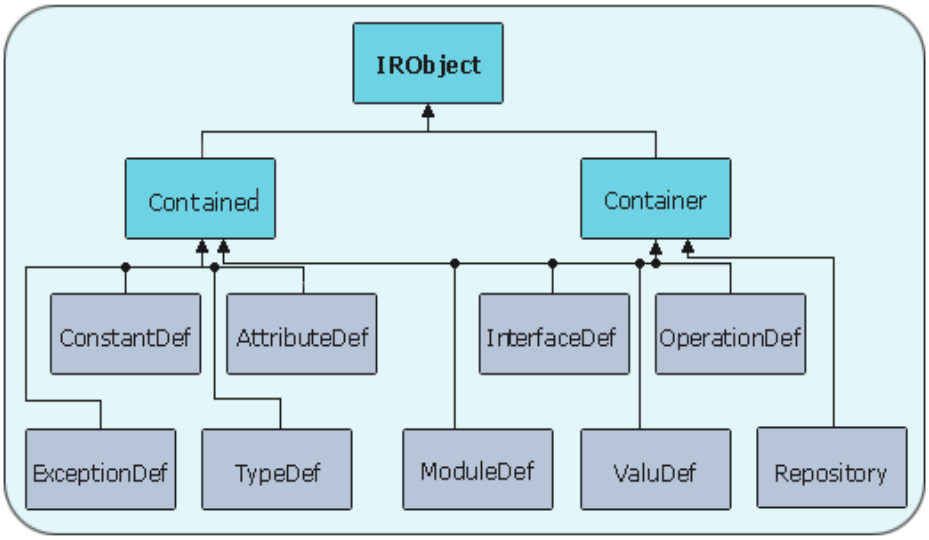
Il cliente riceve i risultati senza accorgersi della modalità dinamica

INTERFACE REPOSITORY

Interface Repository si occupa di **registrare** tutte le **interfacce** e di gestirne la **memorizzazione** e la **ricerca**

(e non degli oggetti specifici che le realizzano)

Il repository si comporta come un **contenitore** con **contenuti**



INTERFACE REPOSITORY in CORBA

Interface Repository non è un sistema di nomi, ma permette di esplorare le interfacce disponibili
consente l'accesso in modo

diretto o attraverso **utilità proprietarie**

Ogni entità viene anche etichettata da un *RepositoryID*

Si raccomandano alcuni formati diversi:

IDL	IDL:/Vai/Servizi/Interfaccia:1.0
RMI hashed	RMI: nome … /hashcode
DCE format	DCE: UUID
Local format	LOCAL: libero

Si standardizzano e raccomandano operazioni di accesso

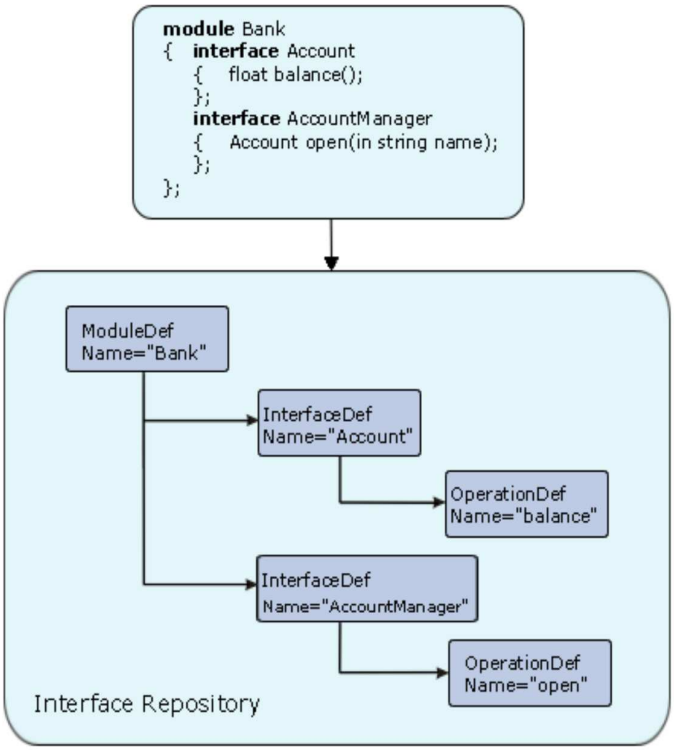
```
Contained lookup_id (in RepositoryID searchid);
InterfaceDef get_interface();
```

INTERFACE REPOSITORY in CORBA

Ad ogni interfaccia definita e compilata

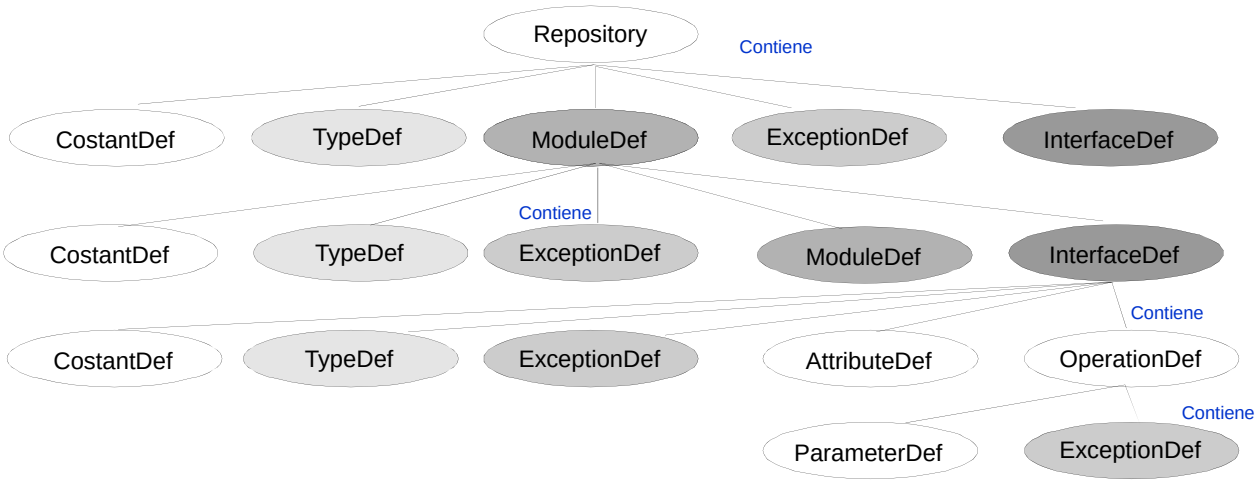
Si generano delle trascrizioni delle informazioni nell'IR

in base ai tipi che possono essere riconosciuti



INTERFACE REPOSITORY in CORBA

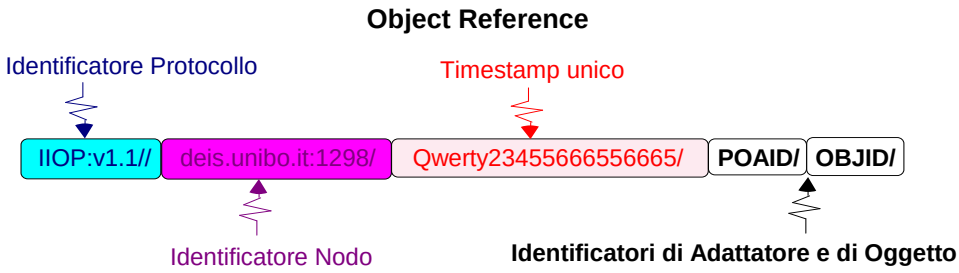
Struttura più completa dei tipi in un IR



RIFERIMENTI AD OGGETTI in CORBA

I **riferimenti** di CORBA sono **opachi** e **permettono di arrivare ad un POA e di trovare un servant** senza garanzie di un **servant specifico**

Tipicamente possono contenere tutte le **informazioni** per arrivare ai servant: **indirizzo, nome del POA di creazione, object ID (*vari dati*)**



Problema delle diverse forme di informazioni anche visibili e su cui l'utente può fare affidamento

RIFERIMENTI AD OGGETTI in CORBA

Gli **identificatori** disponibili ad un **cliente CORBA** sono del **tutto validi solo per l'ambiente di uso e opachi per l'utilizzatore**

Sono completamente distinti dal modo con cui l'ORB tiene conto degli oggetti stessi e li passa da un ambito ad un altro: esistono quindi ***nomi validi solo in alcune località*** e invece altri con la *possibilità di* ***identificare oggetti specifici*** (*servant*)

Un nome utente (*tipico di un ambiente di linguaggio*) al passaggio **viene convertito** per potere essere utilizzato dal ricevente

In un ambiente ricevente il riferimento potrebbe essere anche per un oggetto diverso con la stessa interfaccia

in caso di stato sul server, problemi

E se si volessero identificazioni precise e mirate?

RIFERIMENTI AD OGGETTI in CORBA

Necessità di **identificatori che possano passare tra ambienti diversi mantenendo la identità del servitore (servant target)**

CORBA 1.2 **non prevedeva nomi unici**

Object Reference (OR) come nomi **non unici** associati ad un servizio e **non ad un servant specifico**

> *ObjectRef* al passaggio viene convertito da un sistema di nomi in un proxy nell'ambiente del ricevente per potere essere utilizzato
>
> *ObjectRef* devono potere essere passati da un ambiente ad un altro ed essere utilizzati dovunque ma non necessariamente per lo stesso oggetto

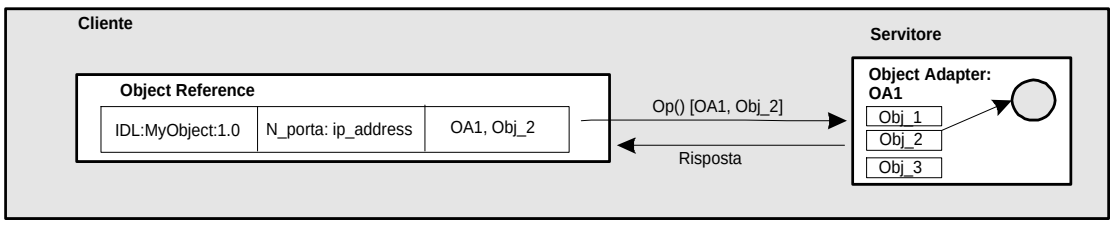
Gli identificatori interni dell'ORB sono spesso legati all'oggetto (servant) specifico

Gli OR non davano garanzia di portabilità

IOR e NOMI UNICI in CORBA

Interoperable Object Reference come nomi unici associati ad un servizio (**IOR**) che possono essere portati tra ORB diversi (*passando anche attraverso stringhe*)

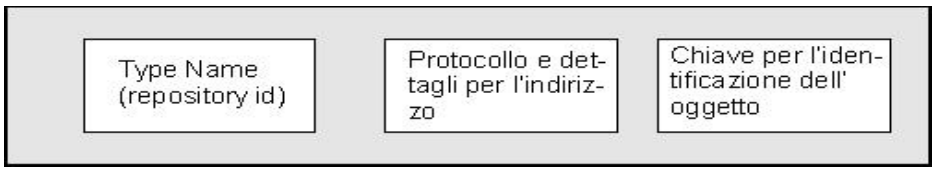
In genere, prima di passare da un ORB ad un altro OR ⇨ **IOR**



CORBA 2 **supporto per nomi unici**

Identificatori unici e mirati a un target specifico

IOR



Interoperable Object Reference o IOR

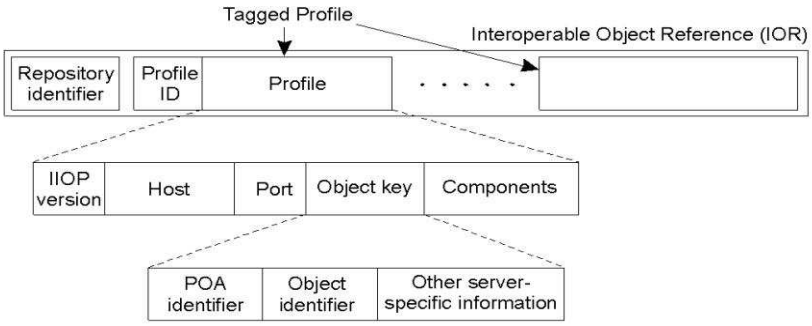
Accordo tra le rappresentazioni dei diversi ORB per i diversi oggetti che esistono al di fuori dell'ORB

Interoperable Object Reference o IOR come standard

IOR come ProfileID (identificatore) e tagged Profile (per determinare completamente)
anche più di un profilo per accessi diversi

Tagged Profile
informazioni complete per la ricerca dell'oggetto

Si usano queste informazioni per decidere cosa passare al cliente che richiede una operazione (poi gli viene fornito il proxy locale per l'oggetto stesso)



IOR in CORBA

Possiamo avere due possibili forme di IOR

Con diversi supporto per arrivare al servant via POA

e supporto dell'Implementation Repository (IMR), che interviene per fornire su bisogno anche la ri-generazione dei servant registrati

Legame indiretto (indirect binding) se **IOR** riferisce il **repository IMR** e solo **indirettamente** attraverso questo l'oggetto finale

Il legame indiretto è quello duraturo e persistente

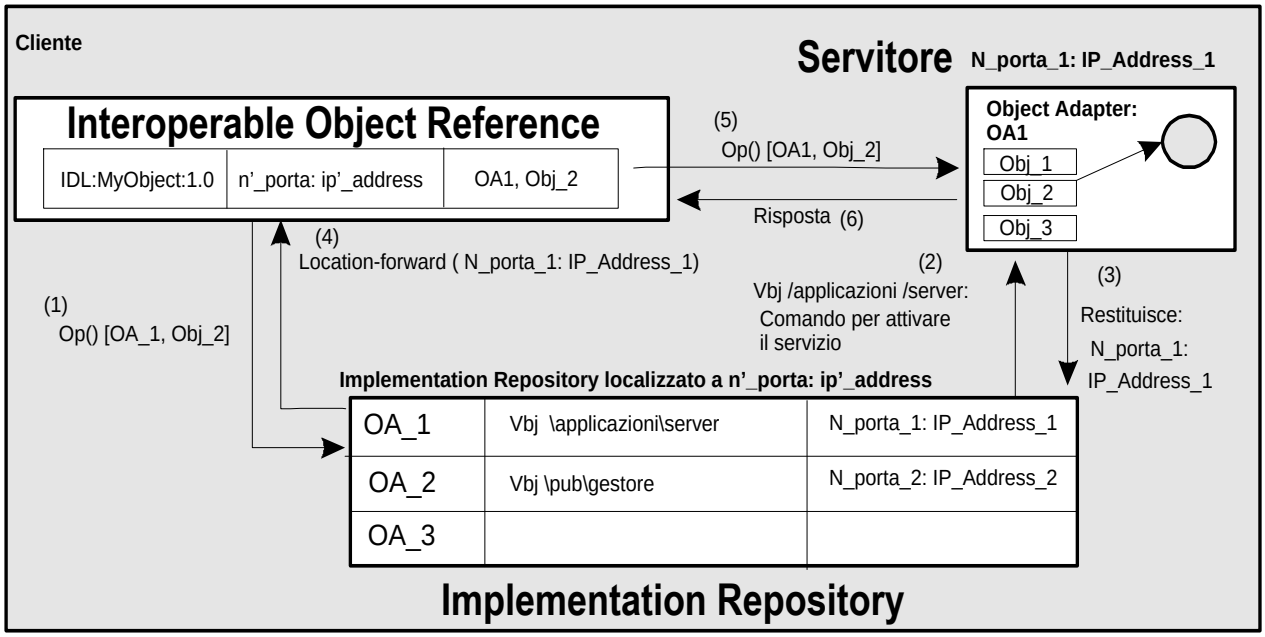
con inserimento in un **Implementation Repository** (al deposito la prima volta), detto **Repository Identifier**

Legame diretto (direct binding) se **IOR** riferisce **direttamente** l'oggetto relativo (via POA)

Il legame diretto per oggetti transienti

IOR (binding indiretto via POA)

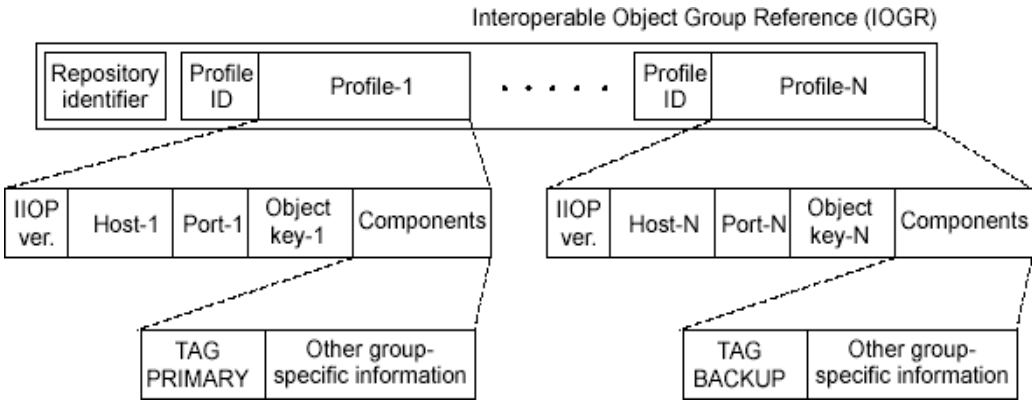
Legame indiretto (indirect binding)



ESTENSIONI: Gruppi di oggetti

CORBA3 prevede la possibilità di associare copie multiple ad un servizio
con forme di replicazione del **tutto trasparente** al cliente

Si introducono **I**nteroperable **O**bject **G**roup **R**eference (**IOGR**) ed è compito dell'ORB di trovare le **copie disponibili** per le funzionalità richieste
(anche gestire disconnessione / riconnessione /consistenza)



ESTENSIONI: PASSAGGIO per VALORE

CORBA prevede di trattare gli oggetti (**fissi in allocazione**) per riferimento

Per avere la possibilità di passare **oggetti non primitivi per valore**, CORBA3 definisce i valuetype

Un *valuetype* permette di ottenere una **nuova copia** di un contenitore riempito con i **valori dall'ambiente mittente** producendo un **nuovo oggetto** dalla parte del **ricevente**

- L'oggetto (il tipo) deve essere ***noto dalla parte del ricevente***
- Oggetti valuetype sono trattati sempre ***localmente*** e non sono registrati all'ORB (costo limitato)
- Le operazioni devono essere ***disponibili localmente*** (nessun contributo di CORBA)
- Al passaggio, si fa ***marshalling dei valori*** e ***unmarshalling*** assumendo consistenza dei tipi

OBJECT ADAPTER in CORBA

Object Adapter come **agenti intermedi** per consentire di superare i problemi della **eterogeneità** dei diversi ambienti

BOA (Basic Object Adapter) come la entità di base
Si permettevano la attivazione dei server prevedendo solo alcune politiche semplici e molte altre politiche del tutto implementation-dependent

Shared server	un'**unica attività** per un insieme di oggetti (attività unica condivisa)
Unshared server	un'**attività** per **ogni servant**
Persistent server	un'**attività unica** fatta partire alla inizializzazione o esplicitamente
Per Method server	un'**attività** per **ogni invocazione**

PORTABLE OBJECT ADAPTER - POA

POA come **agente portabile interoperabile** che permette di passare da un **object reference** di un cliente al **codice concreto** del **servant** che deve servire la richiesta stessa

*Un POA può gestire **molti oggetti diversi** e seleziona su quali **dirigere le operazioni***

In **ambienti diversi**, il **POA è diverso** (classi, variabili, metodi, attività) ma *deve potere realizzare le politiche di base necessarie alla varietà delle interazioni possibili*

In un **ambiente di linguaggio specifico**, esiste *una classe base* da cui derivano tutti i POA e che contiene i meccanismi per la gestione della richiesta e dei servant

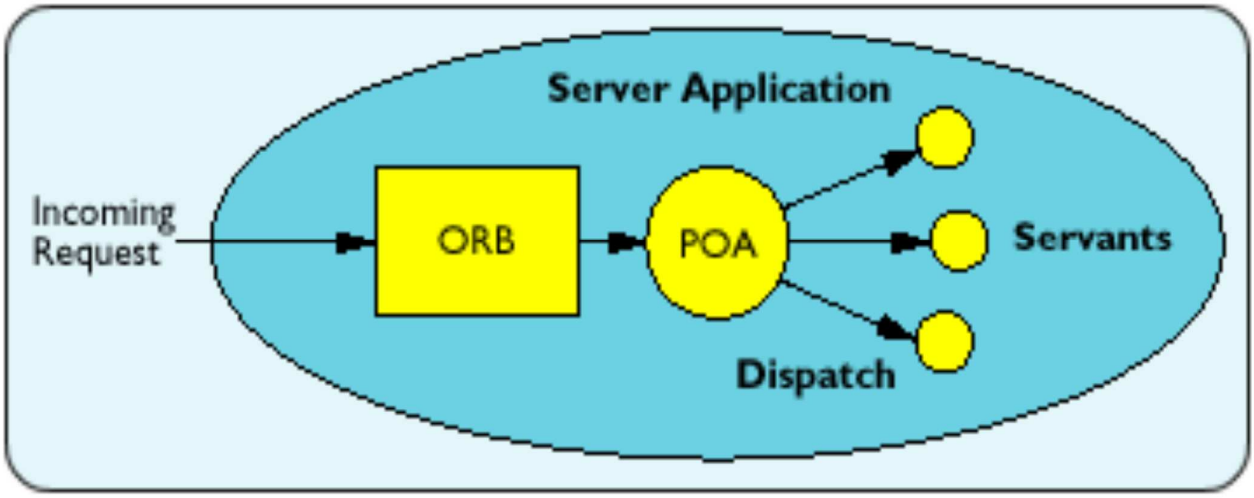
Il *POA non eredita le politiche* definite caso per caso

PORTABLE OBJECT ADAPTER - POA

POA come agente portabile per la interoperabilità

Eredita da **altri POA** (di default) senza ereditare le politiche che devono essere espresse ad-hoc

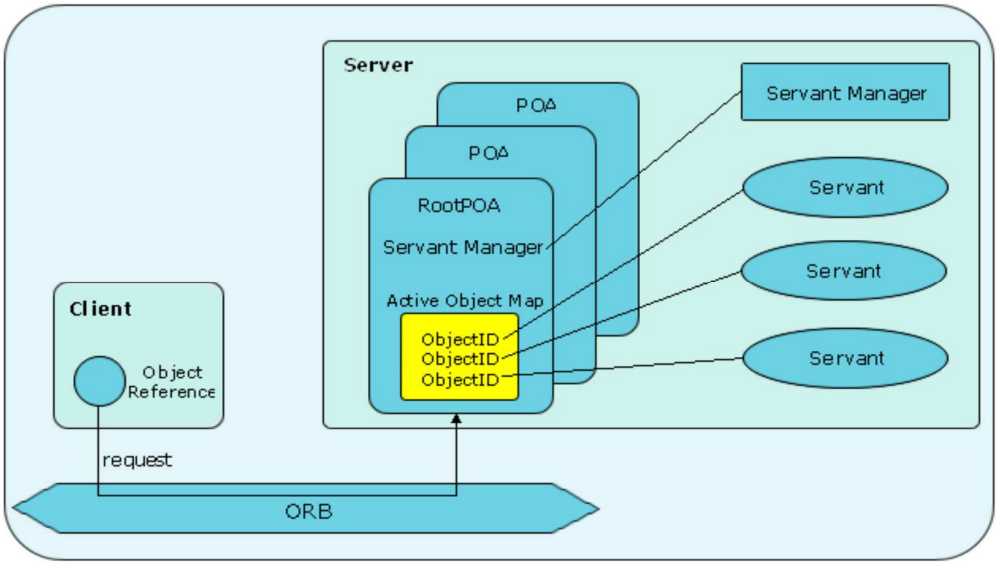
Politiche anche diverse e specializzate



PORTABLE OBJECT ADAPTER

Il POA ha una **organizzazione interna** sua propria (**AOM**)

Una tabella interna, **Active Object Map**, permette al POA di mappare i servant (anche lo stesso molte volte)



PORTABLE OBJECT ADAPTER MANAGER

Il **POA** viene gestito attraverso un **POA Manager** per consentire le **politiche di gestione** (***mappare servant*** *e* ***object reference***). Il **POA Manager** prevede le operazioni per consentire **diverse politiche** e anche **cambiarle**

Il POA Manager permette di:

- **attivare un POA** (per fare partire il lavoro)
- **disattivare un POA** (per chiudere il lavoro dei POA)
- **bloccare le richieste** ai POA (il lavoro viene bloccato e non si fa partire nessuna operazione)
- **scartare le richieste** per i POA (tutte le richieste in arrivo e accodate sono scartate: nessuna operazione)

Su un **POA disattivato** si possono **cambiare politiche**

OBJECT ADAPTER in CORBA

POA come agente portabile per la gestione degli oggetti e dei servant: tipicamente **un POA gestisce con una stessa politica almeno una interfaccia** (spesso **più di una**)

RESPONSABILITÀ di

Creazione Object Reference

Identificazione degli ObjectID (identificatori unici di servant)

Gestire i servant relativi

Oggetti **CORBA transienti**

che non sopravvivono alla applicazione che li ha generati

Oggetti **CORBA persistenti**

che sopravvivono alla applicazione che li ha generati e rimangono disponibili anche per altre successive applicazioni

Funzioni di ADATTATORE

I **POA** hanno dei **metodi** visibili ai clienti per registrare servant

`ObjectID` **`activate_object`** `(in` **`Servant p`**`);`

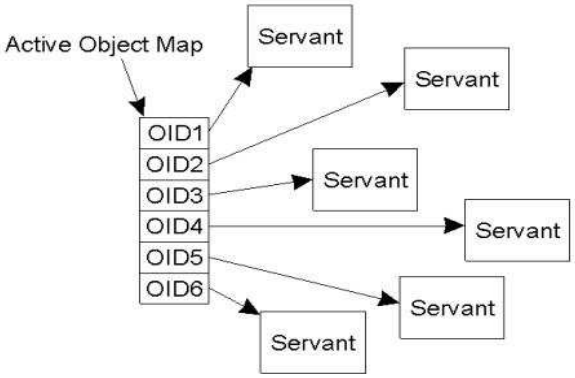
restituisce un identificatore di oggetto e riceve un puntatore a servant

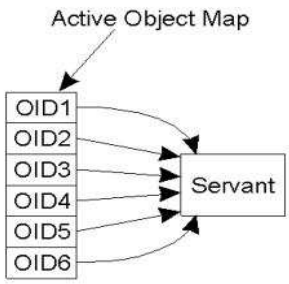
`void` **`activate_object_with_ID`**`(in` **`ObjectID oid`**`, in` **`Servant p`**`);`

associa un puntatore a servant ad una entry nella mappa **Active Object Map**

Il metodo permette anche la scelta esplicita tra servant nella **AOM**

ObjectID permettono la scelta dei servant nel POA





OBJECT ADAPTER in CORBA

POA come agente portabile per la interoperabilità

le **politiche** derivano da **proprietà specificate** ad-hoc con **attributi standardizzati**:

- **Thread** (ORB_CTRL_MODEL, SINGLE_THREAD_MODEL)
- **Lifespan** (TRANSIENT, PERSISTENT)
- **Object ID Uniqueness** (UNIQUE_ID, MULTIPLE_ID)
- **ID Assignment** (USER_ID, SYSTEM_ID)
- **Servant Retention** (RETAIN, NON_RETAIN)
- **Requests** (USE_ACTIVE_OBJECT_MAP_ONLY, USE_DEFAULT_SERVANT, USE_SERVANT_MANAGER)
- **Implicit Activation** (IMPLICIT_ACTIVATION, NO_IMPLICIT_ACTIVATION)

ATTRIBUTI per OA in CORBA

POA prevede **diversi valori** che combinati producono molte possibilità molto differenziate (default in rosso):

- **Thread** (ORB_CTRL_MODEL, SINGLE_THREAD_MODEL)
- **Lifespan** (TRANSIENT, PERSISTENT)
- **Object ID Uniqueness** (UNIQUE_ID, MULTIPLE_ID)
- **ID Assignment** (USER_ID, SYSTEM_ID)
- **Servant Retention** (RETAIN, NON_RETAIN)
- **Requests** (USE_ACTIVE_OBJECT_MAP_ONLY, USE_DEFAULT_SERVANT, USE_SERVANT_MANAGER)
- **Implicit Activation** (IMPLICIT_ACTIVATION, NO_IMPLICIT_ACTIVATION)

Retention e Request Processing Policy

- Retention policy: prevede l'utilizzo o meno dell'AOM
  - **RETAIN**: memorizzazione di tutti gli Object Id nell'AOM
  - **NON_RETAIN**: **NON** si usa **AOM** → uso di Default Servant, o di Servant Manager
- Request Processing policy: indica la modalità di reperimento degli oggetti serventi per l'elaborazione delle richieste
  - **USE_ACTIVE_OBJECT_MAP_ONLY**: il dispatching avviene per gli oggetti servant registrati presso AOM
  - **USE_DEFAULT_SERVANT**: (se è impostata una politica **NON_RETAIN**, oppure l'oggetto servente non è nell'AOM) le richieste per **oggetti serventi** non disponibili nel POA sono delegate ad un'unico **servant**, detto **Default Servant**
  - **USE_SERVANT_MANAGER**: politiche di attivazione/disattivazione degli oggetti servant a carico di un Servant Manager, specificato e gestito **direttamente dall'utilizzatore finale**

POLITICHE PER I POA

POA può consentire politiche molto differenziate per la gestione degli oggetti servant

Politiche a default POA:

Single Servant (per tutti gli oggetti)

Un solo servant per tutte le richieste (anche oggetti di tipo diverso)

Explicit Object Activation

Ogni servant specifico è collegato ad un ObjectID, con molto controllo del servant per la esecuzione del servizio

On-Demand activation (per un singolo metodo) senza stato

On-Demand activation (per durata indefinita)

il servant si attiva su richiesta e si mantiene per ogni richiesta successiva

Le politiche si possono anche combinare

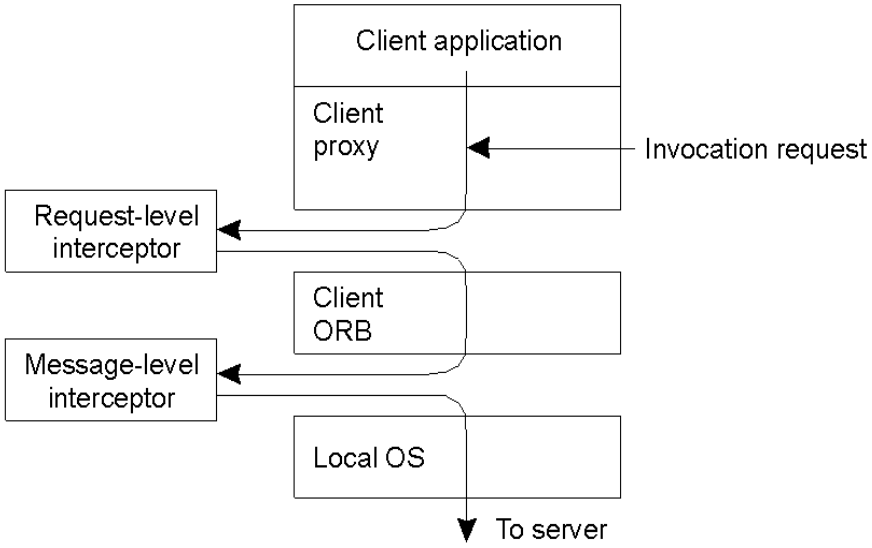
INTERCEPTOR in CORBA

Per aggiungere servizi o funzioni in modo trasparente si introducono interceptor senza cambiare niente del server e del client

Ai diversi livelli del sistema

- applicazione
- trasporto

- per sicurezza
- per transazioni
- …



CORBA 3

CORBA 3 introduce alcune significative aree di estensione / completamento

Internet

nomi come URL, firewall proxy per GIOP, …

QoS

nuove forme di invocazione con maggiore controllo QoS Asynchronous calls (**AMI**) & Time-independent (**TII**)

CORBA Real-time, CORBA ridotto, CORBA fault-tolerant

Componenti

livello più astratto per lavorare in modo trasparente

SEMANTICA di INVOCAZIONE

In CORBA le invocazione sono sincrone

Il cliente deve attendere il completamento della operazione da parte della infrastruttura

Operazioni statiche sempre sincrone (at-most-once)

Operazioni dinamiche anche meno sincrone

one-way *senza risultato* (best-effort)

nessuna risposta prevista al server

deferred-synchronous *risultato ritardato* (at-most-once)

il cliente può successivamente aspettare la risposta che il server deve mettere a disposizione in seguito

ASYNCHRONOUS INVOCATION - AMI

Le invocazioni di CORBA non sono **persistenti e disaccoppiano poco …**

CORBAMessaging introdotto per trattare modi di invocazione non possibili nello standard di base CORBA

Si intendono **disaccoppiare:**

- **la operazione del servant (con risultato normale e sincrono) dalle modalità di invocazione del cliente**
- **il tempo di vita** dei due ambienti

con modalità **Callback** e **Polling**

si **modifica la interfaccia del cliente** e si ottiene di potere movimentare le richieste e di potere avere una **interazione** diversa da quella previste dal server grazie all'ORB …

Il cliente deve **abilitare** le **operazioni aggiuntive**

ASYNCHRONOUS INVOCATION - AMI

Callback: il cliente fornisce un **metodo di callback** richiamato dal supporto al completamento attraverso una specifica **fire-and-forget (invocata automaticamente)**

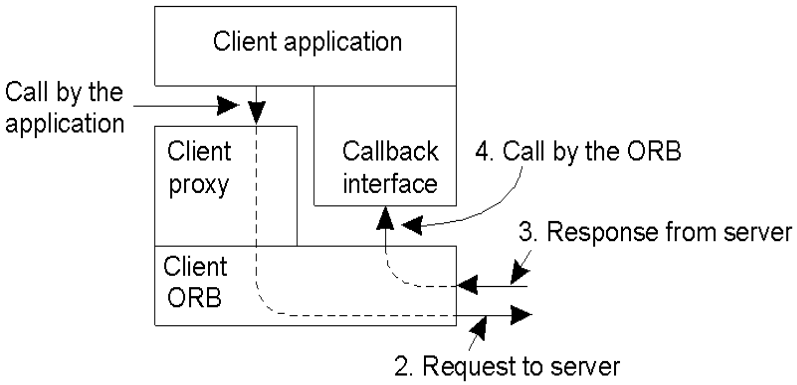
La interfaccia statica viene modificata:

```
int  somma (in int i, in int j, out int somma)
void  sendcallback_somma (in int i, in int j, callbobj)
void  callback_somma (in int success, in int somma)
```

*Usiamo due metodi cambiando solo la **implementazione cliente** e non la parte di servizio*

Cliente chiama **sendsomma**

ORB invoca **callbacksomma specificata dall'utente**



POLL ASYNCHRONOUS INVOCATION

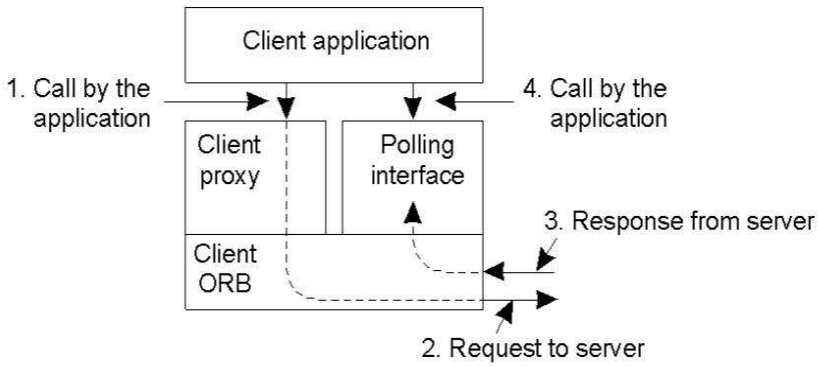
Asincrona polling: il cliente decide **quando e se** interrogare un metodo di verifica del completamento della operazione remota (ottenendo il / i risultati) **creato/i dal supporto**

Anziché: `int  somma (in int i, in int j, out int somma)`

```
void  sendpoll_somma (in int i, in int j, pollobj)
void  pollsomma (out int success, out int somma)
```

Per trattare **polling**

Il cliente invoca la **sendpollsomma** e quando vuole recupera il risultato invocando l'operazione **pollsomma** generata automaticamente dal supporto CORBA



MESSAGING

Possibilità di impostare la QoS per i messaggi

Interfaccia **RebindPolicy** per ripristinare la connessione se caduta

`TRANSPARENT, NO_REBIND, NO_RECONNECT`

Interfaccia **SyncScopePolicy** per stabilire la garanzia di sincronizzazione

`SYNC_NONE, SYNC_WITH_TRANSPORT, SYNC_WITH_SERVER, SYNC_WITH_TARGET`

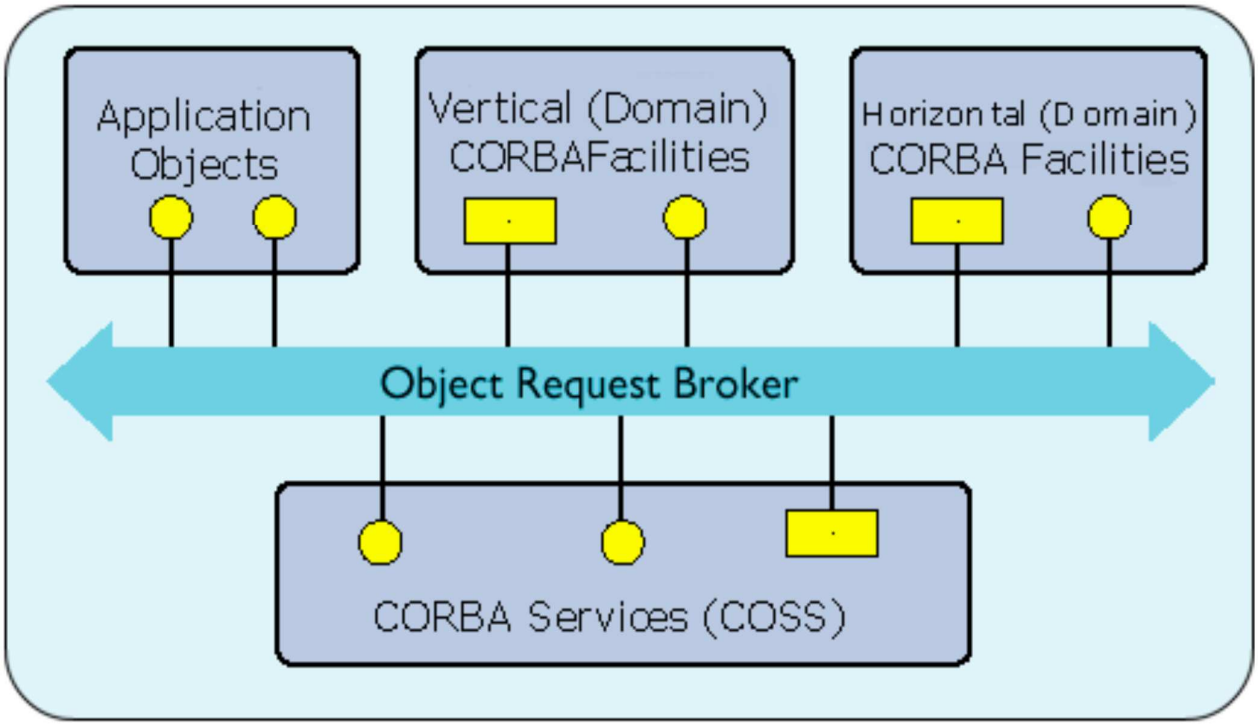
Interfacce **RequestPriorityPolicy** e **ReplyPriorityPolicy** per la priorità delle due parti che compongono invocazione, se necessario

Interfacce **QueueOrderPolicy** per la priorità con cui gestire l'ordine delle richieste

`ORDER_ANY, ORDER_TEMPORAL, ORDER_PRIORITY, ORDER_DEADLINE`

Altre possibilità …

ARCHITETTURA CORBA



CORBA SERVICES

CORBA richiede anche molte altre parti

I **CORBA Services** permettono di fornire funzioni di appoggio per ottenere servizi *più o meno essenziali*

Collection service per raggruppare oggetti

Query service per query per interrogare oggetti

Concurrency (control) service per servizi pronti di lock

Event service per usare eventi asincroni

Notification service gestione avanzata eventi

La presenza di questi servizi qualifica CORBA come un ambiente di integrazione di componenti

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

CORBA SERVICES

OMG ha standardizzato altri componenti per facilitare la programmazione ed il supporto: **è** (in pratica) **necessario** il

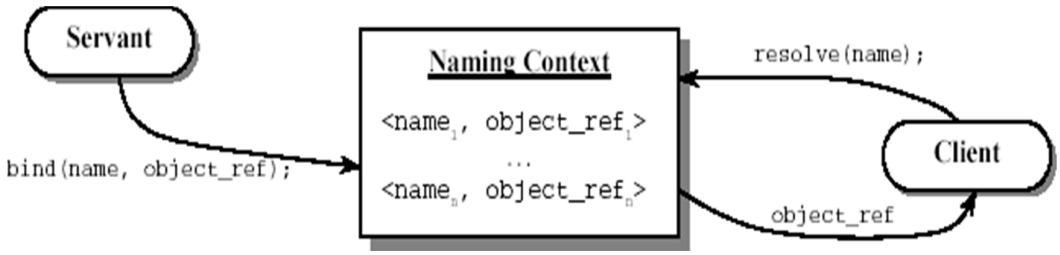
NAMING SERVICE

Meccanismi e strategie di supporto ai **nomi persistenti** di CORBA, per classificare e ritrovare **ObjectReference attraverso nomi logici** e per realizzare sistemi di nomi usabili

Name binding come associazione tra **oggetto** e **nome**

Name context come insieme di binding in cui ognuno dei nomi (delle coppie) è unico

I binding sono per definizione relativi ad un contesto specifico e da specificarsi

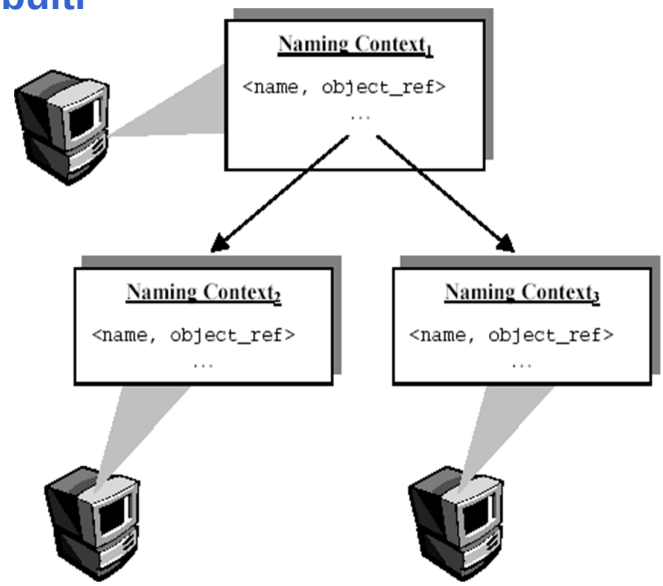


NAMING SERVICE

Un **nome** strutturato come una sequenza di **componenti di nome per identificare l'ObjectRef corrispondente**

Nomi diversi possono fare riferimento a ***oggetti diversi*** o allo ***stesso oggetto*** ritrovandolo con un processo di **risoluzione in diversi contesti anche distribuiti**

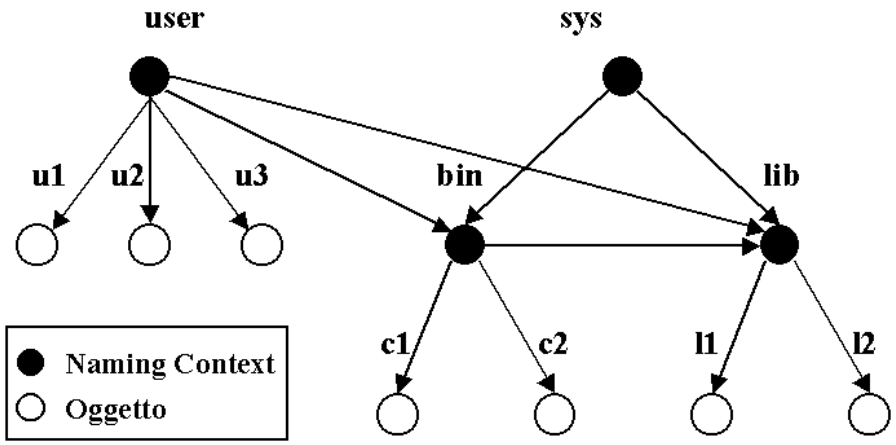
I nomi possono anche fare riferimento a contesti federati con **server federati** (e diversi come contesti da gestire) e **coordinati tra loro**



NAMING SERVICE

Si determinano dei grafi di contesti e di riferimenti agli oggetti da relazioni di contenimento di contesti (attraverso ObjectReference)

I contesti sono trattati come oggetti CORBA



NAMING SERVICE

Un nome **semplice** o **composto** da una **sequenza** di componenti di nome

Ogni **componente** costituito di due parti o attributi

[Identifier , Kind]

Identifier come Object Reference di tipo CORBA Object

Kind di tipo descrittivo, ad esempio *executable, postscript*

```
struct NameComponent {string id; string kind;};
typedef sequence <NameComponent> Name;
```

Si forniscono solo i meccanismi di base su cui costruire le diverse politiche utente

La idea è che il servizio fornisca solo meccanismi e non politiche di nessun tipo

NAMING CONTEXT

Le operazioni che si possono considerare su un contesto di namig derivano dalla Interfaccia NamingContext che richiede le tipiche operazioni di un sistema di nomi

```
interface NamingContext{
void bind(in Name n; in Object obj) raises …;
void rebind(in Name n; in Object obj) raises …;
void unbind(in Name n) …;
void bind_new_context(in Name n)…;
object resolve(in Name n)…;
void list(in unsigned long how_many,
   out BindingList bl, out BindingIterator bi);
}
```


TRADING SERVICE

Il **TRADING Service** ha l'obiettivo di facilitare la **ricerca di servizi che implementano una certa interfaccia attraverso attributi che li caratterizzino** (*funzionalità simili alle **pagine gialle o …***)

Il Trader è un oggetto che **permette di diffondere** la **conoscenza** dei **servizi che si possono richiedere (come nomi logici)**

Il trader permette di ***esporre servizi***

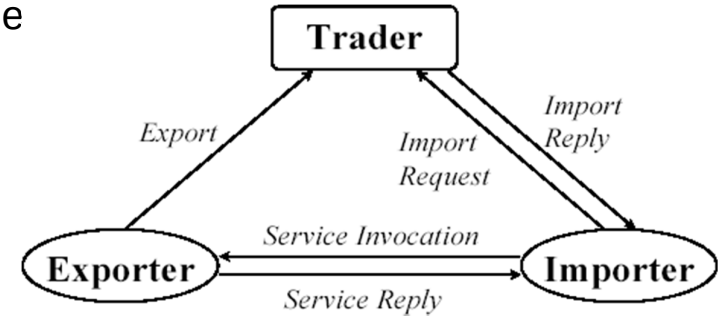
export da parte di chi li fornisce

Il trader permette di ***importare servizi***

import da parte di chi li vuole

Ovviamente avremo anche

Trader federati



TRADING SERVICE

Una **ricerca tipo su un trader** permette di ottenere una **interfaccia non nota (si ottiene il nome)** attraverso la richiesta si **proprietà che la caratterizzano** (**ottenendo anche più nomi**)

CORBA non specifica niente sulla implementazione del **TRADING Service: si possono realizzare con database o anche tabelle in memoria**

Ogni trader è caratterizzato da

- **un'interfaccia che definisce le funzionalità** esposte dal servizio,
- **alcune proprietà** per rappresentare gli aspetti comportamentali e non-funzionali non espressi dall'interfaccia del servizio

Ogni proprietà è identificata da un attributo **PropertyMode**

PropertyMode associata alla tripla <**name, type, mode**>

```
enum PropertyMode {PROP_NORMAL, PROP_READONLY,
 PROP_MANDATORY, PROP_MANDATORY_READONLY}
```


TRADING SERVICE

Interfaccia di un **Servizio associata alle proprietà**

Ogni proprietà è costituita da <**nome, valore**>

con ereditarietà tra servizi (e sulle interfacce considerate)

```
service <ServiceTypeName>
  [:<BaseServiceTypeName> [,<BaseServiceTypeName>]*]
{interface <InterfaceTypeName>;
 [[mandatory] [readonly] property <IDLType>
          <PropertyName>;]*
};
```

La **pubblicazione** avviene fornendo il **nome del servizio**, nessuna o più proprietà, **e** un **nome della/e interfacc-ia/e** che implementa

Con una richiesta otteniamo anche molti nomi

EVENT & NOTIFICATION SERVICE

CORBA prevede **comunicazione sincrona e uno-a-uno**
EVENT SERVICE la rende **più asincrona e flessibile**

Eventi con informazione: **valore** o riferimenti a **oggetto**

Le informazioni possono essere **generiche** o **tipate**

Considerando di base una **conoscenza reciproca di interfaccia**, si considerano **produttori** (supplier) e **consumatori** (consumer) di eventi, con varie modalità di comunicazione

comunicazione diretta o **comunicazione mediata da canali**

e 2 modelli di comunicazione

Modalità **push** i produttori inviano ai consumatori

Modalità **pull** i consumatori chiedono ai produttori (on need)

EVENT & NOTIFICATION SERVICE

CORBA prevede una gestione degli eventi o diretta o anche attraverso **event channel,** come intermediari facilitatori

Se un consumatore non si è registrato al canale e lo fa in un certo istante, ***tutti gli eventi precedenti possono essere persi***

Ogni consumatore registrato *riceve* ***tutti gli eventi*** *che si stanno verificando*

Gli eventi sono **non persistenti**

non affidabili

senza possibilità di fare filtraggio

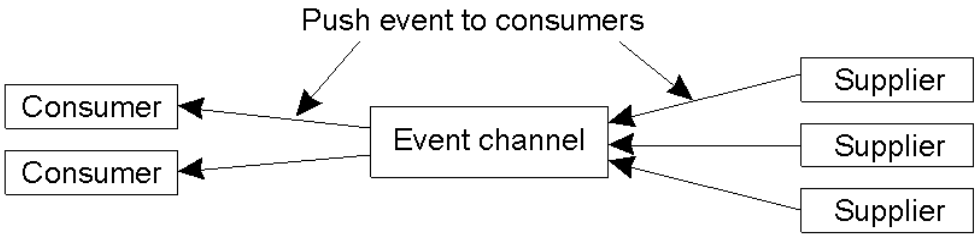
Ma introducono un cambiamento di modello di comunicazione
Proposte per introdurre affidabilità e di notification per filtraggio

INTERFACCE EVENT: Modalità push

In modalità **push** i *consumatori* e *produttori* si conoscono in modo diretto o mediato (con i canali) e si usano le interfacce

```
interface PushConsumer                    {
  void push (in any data) raises(Disconnected);
  void disconnect_push_consumer();  };
interface PushSupplier                    {
  void disconnect_push_supplier();  };
```

le disconnect possono anche terminare e bloccare la comunicazione

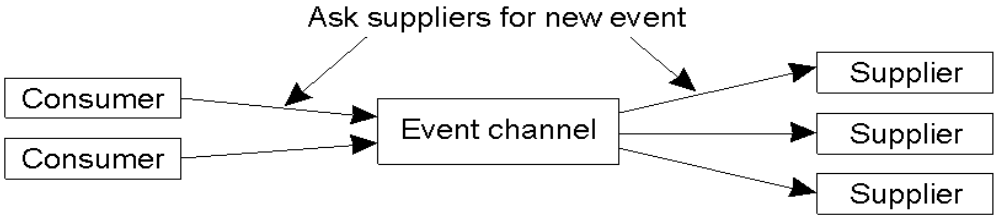


INTERFACCE EVENT: Modalità pull

In modalità **pull** i consumatori e produttori si conoscono tramite le interfacce (o via canali)

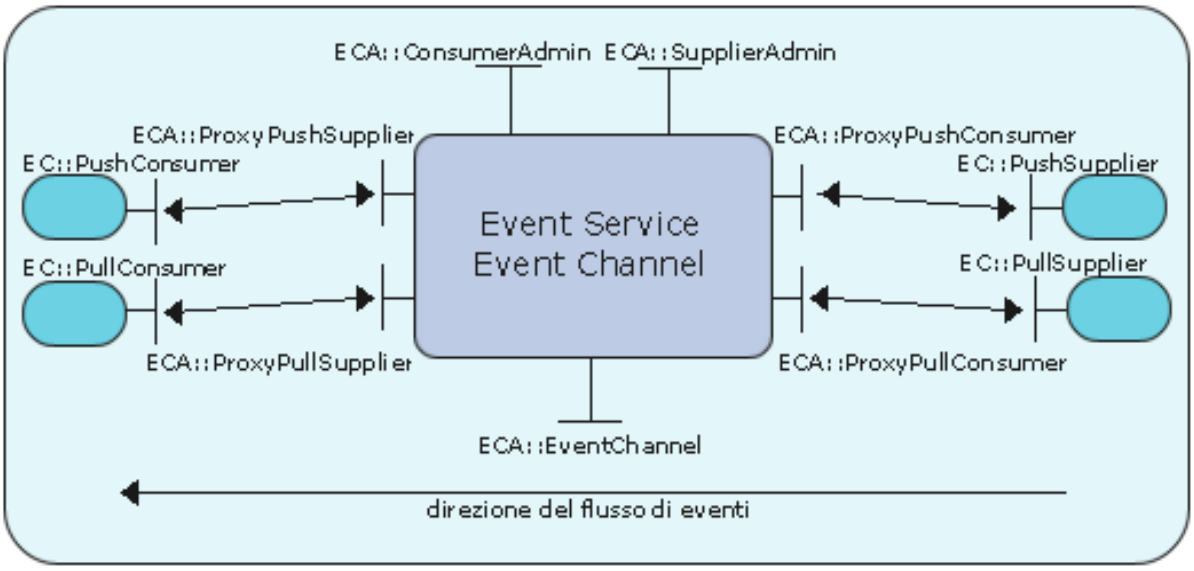
```
interface PullSupplier                    {
  void pull () raises(Disconnected);
  void try_pull (out boolean event)
                    raises(Disconnected);
  void disconnect_pull_supplier();  };
interface PullConsumer                   {
void disconnect_pull_consumer();  };
```

Operazioni di disconnect possono terminare la operatività



EVENT & NOTIFICATION SERVICE

Event Channel oggetto per la comunicazione molti-a-molti



EVENT SERVICE: LIMITI

Event Channel permette e agevola
la **comunicazione molti-a-molti**

Il **Channel** ha la capacità di coordinare anche *più possibili eventi di supplier* prima di scatenare *eventi multipli* su diversi consumer, ma

- **non introduce filtraggio** sui destinatari
- non si occupa di una **qualità di servizio** nella comunicazione delle informazioni (non mantenute stabilmente, permanentemente, …, in dipendenza dalla implementazione specifica)

Il **Notification service** estende il servizio degli eventi con queste molteplici nuove funzionalità

descrizione degli eventi e informazioni, filtri e repository di filtri

NOTIFICATION SERVICE

Gli eventi possono essere caratterizzati da proprietà che ne permettono la ricerca (**Header** e **Body** su cui si possono filtrare)

Reliability (best-effort, persistent), **Priority**, **StartTime**, **Stoptime**, **Timeout**

Si prevede anche un **repository** di **Event type** per la descrizione degli stessi