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

Corso di

Reti di Calcolatori M

MIDDLEWARE VARI

Antonio Corradi

Anno accademico 2014/2015

MIDDLEWARE

Il termine MIDDLEWARE con significato immediato

Insieme degli strumenti che stanno in mezzo tra applicazione e il resto

hardware, **sistema operativo** locale, **tecnologia** locale, …

Il termine **middleware** risale nella storia al **1968**, ad una famosa scuola **NATO** su **Software Engineering**

In ogni caso, il termine è rimasto **poco connotato** fino agli anni 1990, quando i **sistemi distribuiti** hanno cominciato a diventare molto comuni

Middleware associato alla necessità di fare progetti in **sistemi complessi, distribuiti, fortemente eterogenei** e anche per **organizzazioni molto diverse e per servizi molto differenziati**

MIDDLEWARE

Definizione di MIDDLEWARE

Insieme degli strumenti che permettono di integrare **diverse** applicazioni e servizi da utilizzare in **ambienti aperti (eterogeneità) con un ciclo di vita senza limite, pari alla vita della organizzazione**

Middleware comprendono e integrano gli **strumenti di supporto,** per **controllo** e **gestione del servizio durante l'esecuzione**, a tutti i livelli di sistema, dal livello fisico fino al livello di applicazione

RPC middleware (RMI)

Chiamate di Procedura Remote come strumento di invocazione di livello applicativo tra ambienti diversi

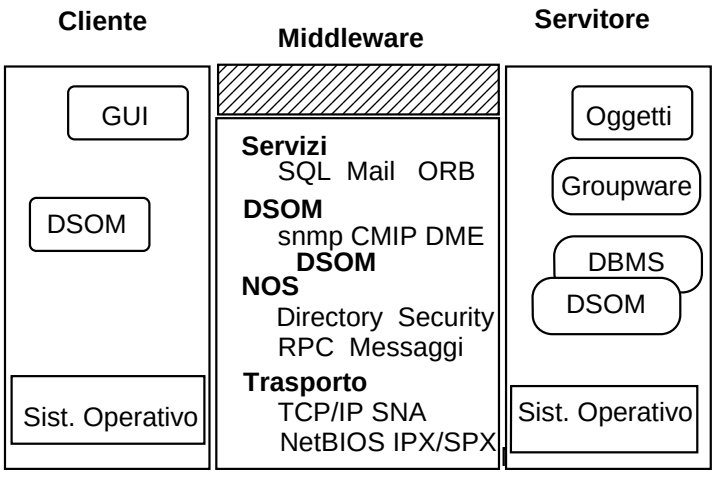
Interazioni tra sistemi e utenti finali (B2B e anche B2C)

MIDDLEWARE: ETEROGENEITÀ

Possibile infrastruttura per la interazione tra ambienti diversi che supera i problemi introdotti da **approcci ad-hoc**

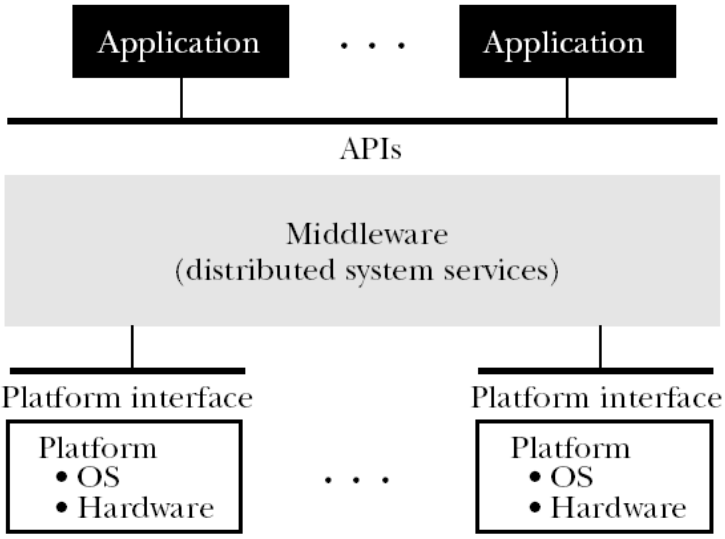
- funzioni di conversione custom
- conversione in formato generico
- wrapper
- protocollo comune

DSOM sistema di gestione distribuita IBM
DME sistema di gestione standard Open Sw Foundation
SNA architettura di rete IBM
IPX/SPX architettura di rete Xerox
…



MIDDLEWARE

Lo **Strato di software** che risiede tra la **applicazione** e i **componenti di rete,** di **sistema operativo locale**, di **hardware eterogeneo**, di **aree di applicazioni diverse**, ecc. per garantire la **corretta operatività**

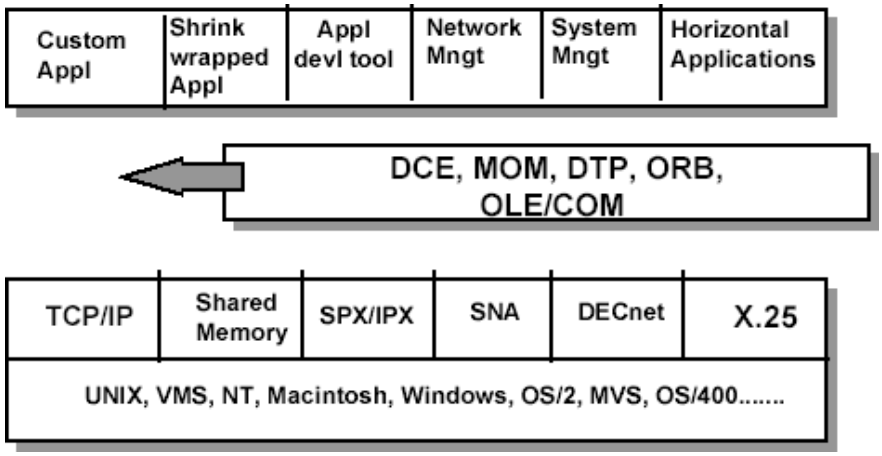


Lo strato di disaccoppiamento tra i livelli di sistema consente un **progetto continuo semplificato** della **parte applicativa** (e anche **del supporto**) e il **superamento della eterogeneità**

MIDDLEWARE

Il Middleware è presente per una organizzazione e consente continuità, permettendo invocazioni **trasparenti** e **accesso uniforme a funzioni locali eterogenee**

- Spesso viene usato per **integrare sistemi legacy** preesistenti e necessari **alla logica aziendale**
- Spesso si propone come **uno standard** (di fatto o di comitato) **per una comunità specifica**



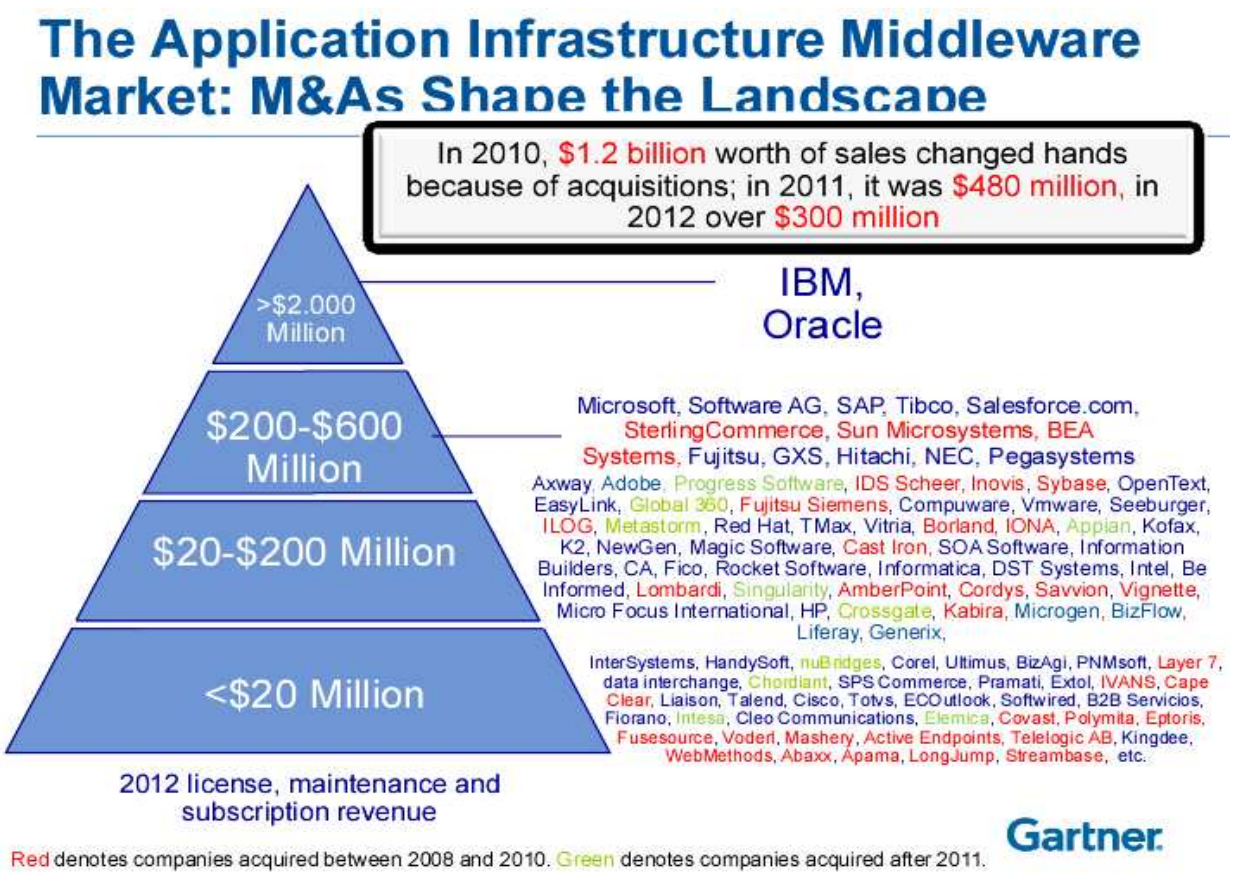
DIFFUSIONE MIDDLEWARE

I Middleware sono ancora un mercato in diffusione in termini quantitativi, secondo Gartner

Una crescita mondiale del **16.4 %** dal 2006 al 2005
Nel 2008 crescita del **6.9 %**, nel 2009 crescita del **2,8** %
Nel 2010 crescita del **7,3** %, fino a **17,6** miliardi di dollari
Nel 2011, crescita del **9,9 %,** fino a **19,3** miliardi di dollari
Nel 2012, crescita del **5,2 %,** fino a **20,3** miliardi di dollari
Nel 2011, crescita del **7,8 %,** fino a **21,9** miliardi di dollari
Nel 2012, crescita del **7,9 %,** fino a **23,8** miliardi di dollari
La suddivisione delle vendite percentuali:

Compagnia	2006%	2007%	2008%	2009%	2010%	2011%	2012%	2013%	2014%
IBM	31.8	28.8	30.8	31.5	32.6	**32.1**	**30.9**	**29.8**	**29.1**
BEA Systems	10.5	9.3	2	oracle	oracle	oracle	**oracle**	**oracle**	**oracle**
Oracle	8.6	8.5	13.6	16.7	17.2	**16.8**	**16.1**	**14.8**	**13.8**
Microsoft	4.2	3	3.6	3.9	4	**5**	**5**	**5.1**	**4.9**
Software AG	2.4	2.2	2.9	3	2.8	**3.3**	**3.2**	**2.7**	**2.3**
Tibco	3.5	3	2.9	2.6	2.5	**2.9**	**2.6**	**2.5**	**2.4**
SAP						**1.2**	**2.7**	**2.4**	**2.2**
Salesforce						**0.7**	**1.6**	**2.2**	**3.1**
Altri	39	45.2	44.2	42.4	40.9	**38**	**37.9**	**40.5**	**42.2**
totale	100	100	100	100	100	**100**	**100**	**100**	**100**

SITUAZIONE MIDDLEWARE 2010-12



FUNZIONI di un MIDDLEWARE

MIDDLEWARE tendono a posizionarsi sopra al sistema operativo e a integrare i servizi, offrendo una serie di funzionalità per

Nascondere la distribuzione

rendere trasparente la applicazione composta di parti in esecuzione su macchine diverse

Nascondere la eterogeneità

rendere trasparente applicazione composta su diversi hardware, per diversi sistemi operativi, usando protocolli diversi …

Fornire interfacce comuni

la applicazione può essere composta anche da parti già fatte, riusata per componenti o sottoparti, trasportata e a massima interoperabilità

Fornire alcuni servizi di base

fornire alla applicazione un ampio repertorio di molte funzioni generali in modo da facilitare la collaborazione ed evitare duplicazione

Garantire la continua disponibilità dei servizi necessari

SERVIZI di un MIDDLEWARE

MIDDLEWARE forniscono servizi differenziati
In senso esteso per molte aree diverse

Presentation Management	(stampa, grafica, GUI, interazione)
Computation	(procedure comuni, servizi caratteri, internazionalizzazione, sorting)
Information Management	(file manager, record manager, database manager, log Manager, …)
Communication	(messaging, RPC, message queue, mail, electronic data interchange…)
Control	(thread manager, scheduler, transaction manager, …)
System Management	(accounting, configuration, security, performance, fault management, … event handling)

MIDDLEWARE come livelli di sistema

MIDDLEWARE tra Applicazione e Sistema Operativo

Applicazione

Domain-specific Middleware Service

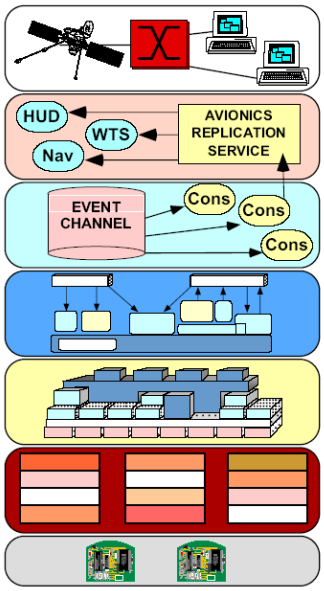
Common Middleware Services

Distribution Middleware

Host Infrastructure Middleware

Sistema Operativo

Hardware

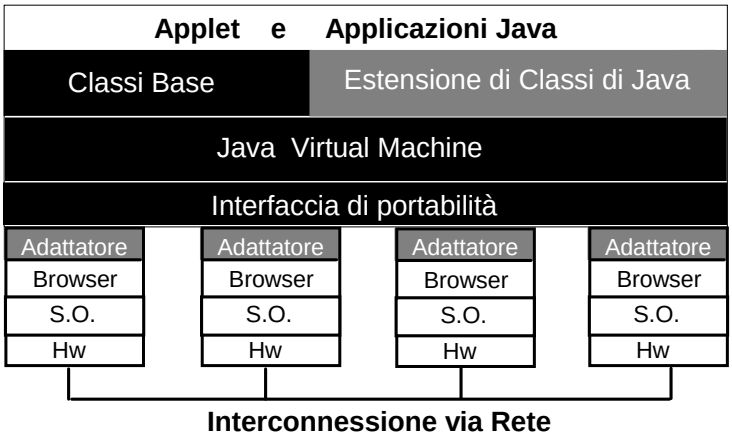


MIDDLEWARE a livelli: Host Infrastructure

Host Infrastructure Middleware

Livello che incapsula e prepara tutti i servizi locali per la loro distribuzione e per facilitare la comunicazione

Esempi: JVM, .NET, altri modelli locali che omogeneizzano costruendo sull'hw e sistemi operativi specifici



Si forniscono delle **API** per arrivare ad avere un supporto locale unificato nei diversi sistemi

MIDDLEWARE a livelli: Distribution

Distribution Middleware

Livello che fornisce il supporto alla programmazione distribuita e facilita le applicazioni, configurando e gestendo le risorse distribuite
Questo livello definisce e crea il modello del middleware

Esempi: RMI, CORBA, DCOM, SOAP, …

Sistemi che permettono una più facile **comunicazione** e **coordinamento** dei diversi nodi che partecipano al sistema introducendo un **modello delle risorse** e

- API di **comunicazione** secondo un modello concettuale
- altre funzionalità di base per la **comunicazione**
  supporto di nomi, di discovery, …

MIDDLEWARE a livelli: Common Services

Common Middleware Services

Servizi aggiuntivi di più alto livello per facilitare il compito del progettista applicativo e per consentire una programma-zione orientata ai componenti fornendo supporto

Esempi: CORBA Services, J2EE, .NET Web Services

Alcune funzioni addizionali per consentire **operazioni facilitate (logica di servizi orizzontali)** ispirate ad una **architettura comune** ed ad un **modello di supporto**

Molti servizi addizionali per componenti utili per tutte le applicazioni: ed esempio, eventi, logging, streaming, sicurezza, fault tolerance, …

MIDDLEWARE a livelli: Specific Services

Domain-specific Middleware Service

Insieme di livelli applicativi dedicati a domini diversi

Esempi: Alcuni gruppi di lavoro stanno definendo **funzionalità ad-hoc per settori diversi** (**logica verticale**) con obiettivi anche molto differenziati di standardizzazione

Task Force di lavoro in **ambito OMG**
- Electronic Commerce TF,
- Finance (banking and insurance) TF,
- Life Science Research Domain TF,

Syngo Siemens Medical Engineering Group,

Boeing Bold Stroke basato su CORBA (trasporto aereo),

…

CLASSI di MIDDLEWARE

MIDDLEWARE

RPC / RMI middleware
Message Oriented Middleware (MOM)
Distributed Transaction Processing (TP) Monitor
Database Middleware
Distributed Object Computing (DOC) Middleware
Adaptive & Reflective Middleware

Altri special purpose:

Mobile & QoS Multimedia Middleware
Agent-based Middleware

Vedi: **computingnow.computer.org**

UN PRIMO MIDDLEWARE !?

Wide Area Distributed Middleware (Web)

Middleware che permette l'accesso in lettura e anche scrittura ad un insieme globale di informazioni

Le operazioni avvengono per un sistema globale

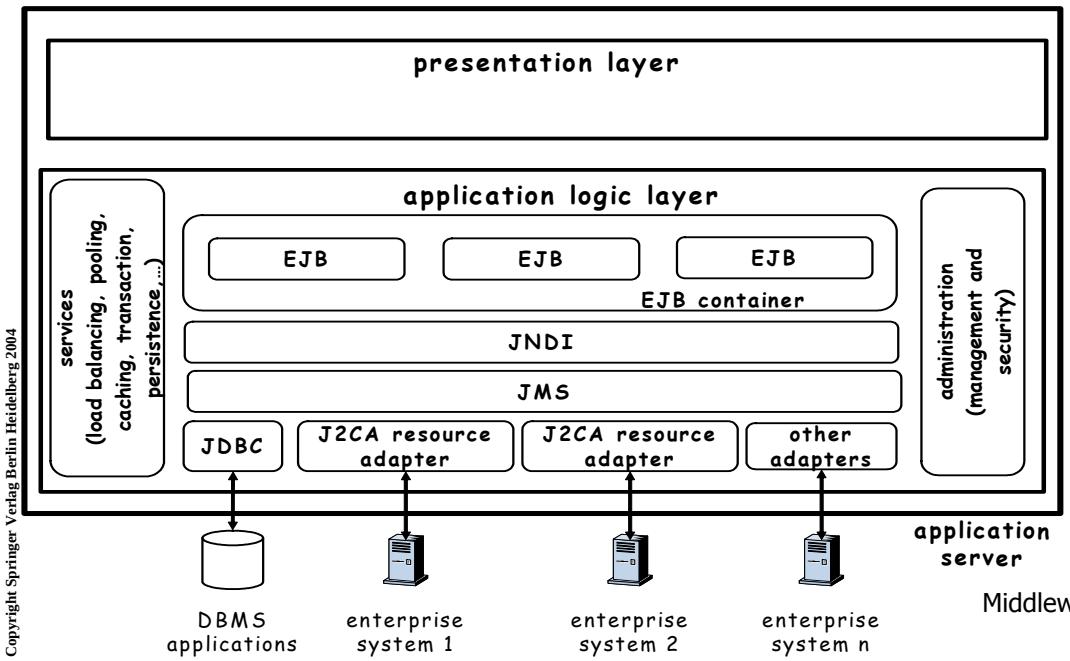
- Moltissimi domini amministrativi di gestione
- Moltissimi utenti
- Moltissimi host partecipanti e
- Molta eterogeneità di banda, interconnessione, **…**

Questo non è un middleware ma forse solo un sistema globale di accesso e messa in linea di informazioni

Web come esempio abilitante e principe di estrema diffusione

WEB MIDDLEWARE !? E J2EE

Visione Java-oriented **J2EE**: **J**ava **N**aming & **D**irectory **I**nterface, **J**ava **M**essage **S**ervice, **J2**EE **C**onnector **A**rchitecture

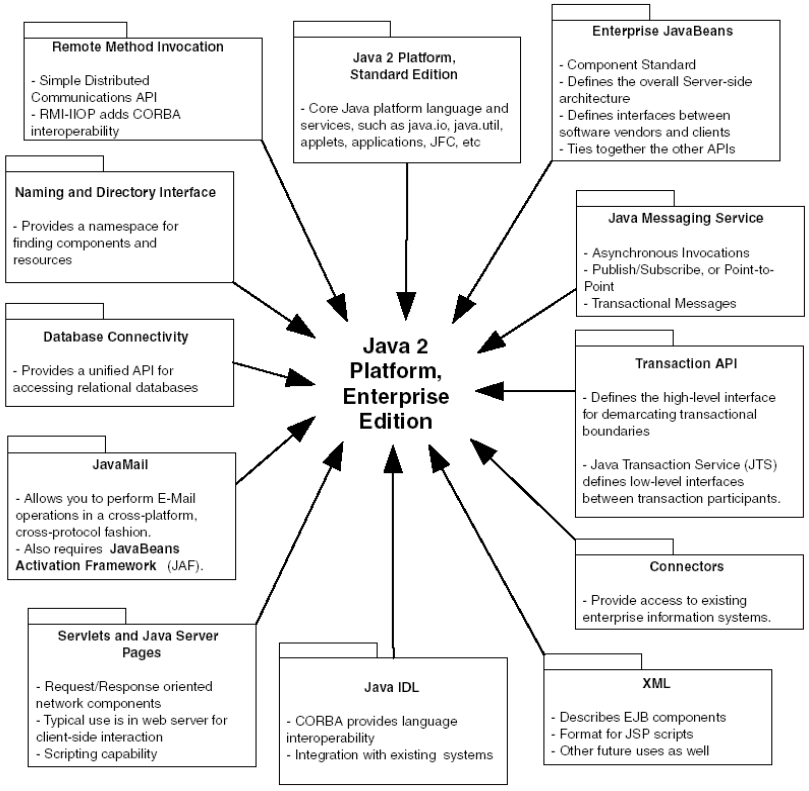


J2EE integrazione di moduli

Java2 Enterprise

insieme di componenti

- database e oltre
- sistemi di nomi
- componenti (Beans)
- integrazione di componenti via XML
- sistemi di messaggi
- comunicazione
- JSP e servlet
- …



PRIMO: RPC MIDDLEWARE

RPC come strumenti per il C/S senza orientamento ad un linguaggio

Modello **rigido, poco scalabile e replicabile** con **QoS**

Il **server deve essere presente** e **prevedere i processi necessari in modo esplicito**

Non tiene conto di eventuali ottimizzazioni nell'uso delle risorse

- **I**nterface **D**efinition **L**anguage (IDL) per l'accordo
- **Sincronicità**: il cliente bloccato in modo sincrono bloccante in attesa della risposta dal Server
- **Gestione eterogeneità** dei dati
- **Uso di Stub** per la Trasparenza
- **Binding** spesso statico (o poco dinamico)

Non troppo flessibile, in crescita via estensione di servizi

DCE (Distributed Computing Environment e simili)

DTP MIDDLEWARE

Distributed Transaction Processing (TP-) Monitor

Middleware per esprimere e supportare **transazioni distribuite**

I **TP monitor** ottimizzano le **connessioni con database** nascondendo decisioni applicative per ottenere accesso ai dati in modo coordinato e transazionale

- interfaccia specializzata **per query** per **clienti leggeri**
- azioni **standardizzate** e **linguaggi ad-hoc** per il settore
- **applicazioni multilivello** con uso di RPC flessibili (variazioni oltre la semantica solo sincrona)
- facilità nel fornire proprietà **ACID**
- capacità di efficienza nel **settore affrontato**

Esempi: **CICS** (**IBM**), **Lotus Notes**, **Tuxedo** (**BEA**), **…**

DB MIDDLEWARE

Database Middleware

Questi middleware nascono per la **integrazione** e l'utilizzo facilitato delle **informazioni contenute in DB eterogenei e diversi**

nascondono dettagli e usano interfacce standard tipo standard **Open DataBase Connectivity ODBC**

- **non** si modificano **DB esistenti**
- **azioni efficienti** (senza troppo accento sulla ottimizzazione e transazioni) in termini di **accesso ai dati**
- **operazioni solo sincrone e standard**
- evoluzione verso il **data mining**

Esempi: **Oracle Glue, OLE-DB Microsoft**

MOM MIDDLEWARE

Message Oriented Middleware (MOM)

La distribuzione dei dati e codice avviene attraverso lo **scambio di messaggi** tra **logicamente entità separate**

Scambio messaggi tipati e **non tipati** sia **sincrono** sia **asincrono** con **strumenti ad-hoc**

- **ampia autonomia** tra i componenti
- **asincronicità e persistenza** delle azioni
- **gestori (broker)** con politiche diverse e QoS diverso
- facilità nel **multicast, broadcast, publish / subscribe**

Esempi: Middleware basati su messaggi, code e gestori **MQSeries IBM**, **MSMQ Microsoft**, **JMS** SUN

OO e DOC MIDDLEWARE

Distributed Object Computing (DOC) Middleware

La distribuzione dei dati e codice avviene attraverso richieste di operazioni tra clienti e servitori remoti

Uso di oggetti come **framework** e di un **broker** come intermediario nella gestione delle operazioni

- **il modello ad oggetti** semplifica il progetto
- **il broker** fornisce i servizi base e molti servizi addizionali
- **alcune operazioni** si possono fare in modo **automatico**
- **la integrazione** di sistemi è **facilitata e incentivata**
- **la tecnologia open source** è favorita

Esempi: **CORBA, COM e .NET, Java Enterprise**

MIDDLEWARE - ANCORA

Adaptive & Reflective Middleware

Middleware che si possono adattare alla **applicazione specifica** anche in modo **dinamico, reattivo e radicale**

*In alcuni casi la **visibilità dei livelli sottostanti** può diventare fondamentale per **raggiungere ottimizzazione***

- **variazioni statiche** dipendenti dal **componente**
- **variazioni dinamiche** dipendenti dal **sistema**

Con la **riflessività**, le **politiche di azione sono espresse e visibili** nel **middleware stesso** e si possono cambiare come **componenti del sistema**

Si raggiunge **adattamento e flessibilità** durante **l'esecuzione**

Esempi: **ancora non molti diffusi**

MIDDLEWARE SPECIALIZZATI

Middleware di supporto alla mobilità

con **componenti pensati per favorire la allocazione e riallocazione trasparente** (lavorando a livello di rete fino al livello di applicazione)

Middleware di supporto a interazioni enterprise

con **componenti** pensati per risolvere i problemi **di servizi enterprise** e supportando **modelli di business specifici**

Middleware per applicazioni e servizi Real-Time

con **supporto** che tende **a garantire tempi di risposta** e **deadline per progetti di servizi tipici del settore RT**

Middleware di supporto a reti ad-hoc

con **componenti e algoritmi molto leggeri,** per **risorse** con **forti limiti di consumo** e **poche capacità**

MIDDLEWARE ENTERPRISE

MIDDLEWARE per fornire servizi di tipo business

Enterprise Application Integration (EAI)

Necessità di facilitare la **integrabilità** tra **strumenti di impresa esistenti** e la loro ampliata applicabilità e disponibilità in azienda

EAI come ambienti per la **integrazione veloce e precisa** di applicazioni e sottosistemi esistenti legacy

Anche interfacce di **rapida prototipazione** di nuove aggregazioni

Sistemi per garantire **più facili operazioni aziendali (in ambiti diversi)**

gestione aziendale (funzioni amministrative e gestionali: SAP)

gestione IT e risorse (funzioni di sviluppo e supporto applicazioni: Websphere, Oracle)

Service Oriented Architecture (SOA)

TINA-C – Middleware per TLC

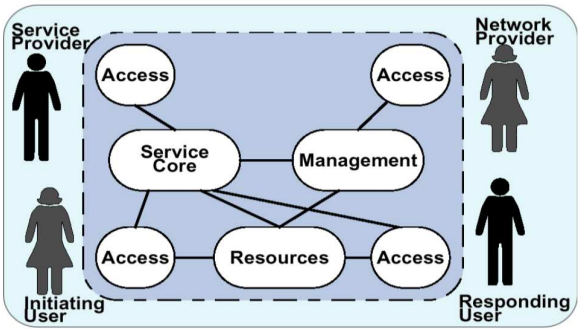
Telecommunications Information Networking Architecture

TINA-C introduce una molteplicità di soggetti coinvolti nel servizio di comunicazione

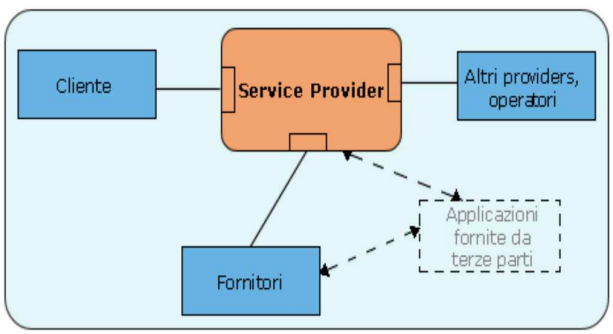
Utenti e molti Provider di comunicazione e servizi

tenendo conto della **qualità di servizio** da fornire **(dopo averla negoziata)**

visione utente

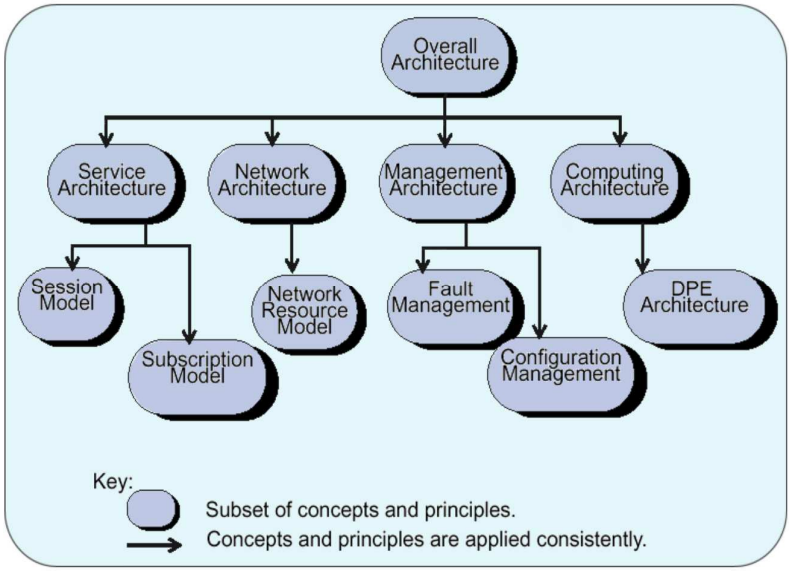


visione interazioni



TINA-C – architetture

Architetture fondamentali **separate interagenti**:
Computing, **Service**, **Management, Network Architecture**



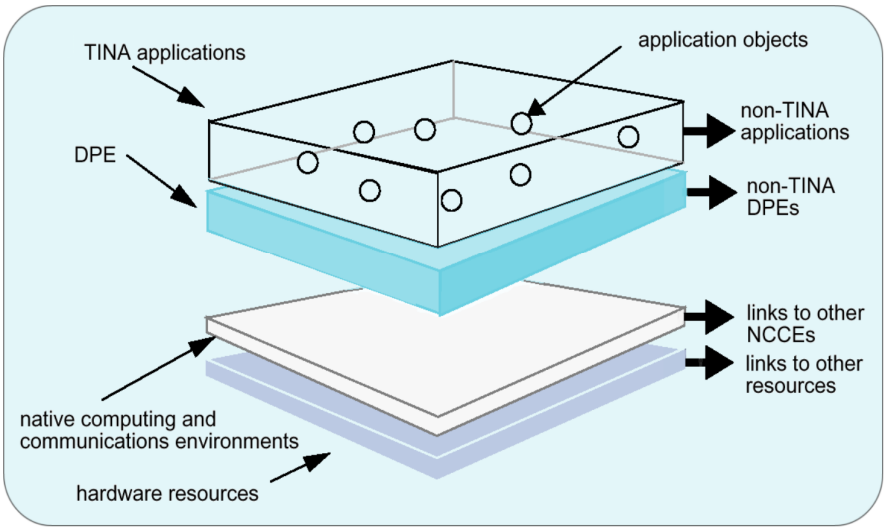
Ovviamente ci sono interazioni tra le diverse architetture

Così come ci sono degli obiettivi comuni di gestione

TINA-C – architetture

In una **visione architetturale** a partire dalla rete

Un nodo deve prevedere di avere delle funzionalità che lo estendono per partecipare al sistema distribuito

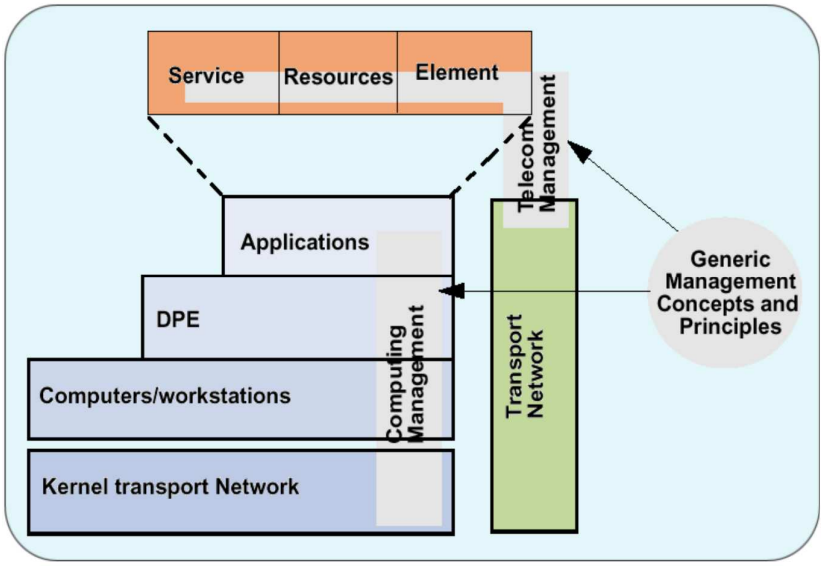


DPE **D**istributed **P**rocessing **E**nvironment

NCCE **N**ative **C**omputing **C**ommunication **E**nvironment

Architettura trasparente

Le **applicazioni** ed i **servizi** sono ottenute al di sopra delle risorse fisiche messe a disposizione dei diversi supporto locali (NCCE) e integrate dal livello DPE

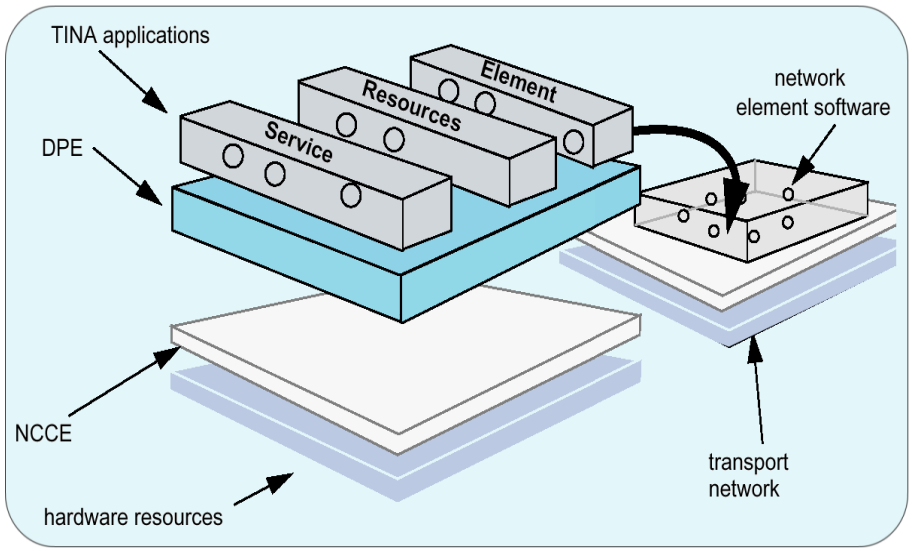


Una applicazione è basata su entità logiche

- **Servizi**
- **Risorse**
- **Elementi**

Architettura trasparente

Visione trasparente delle applicazioni e dei servizi

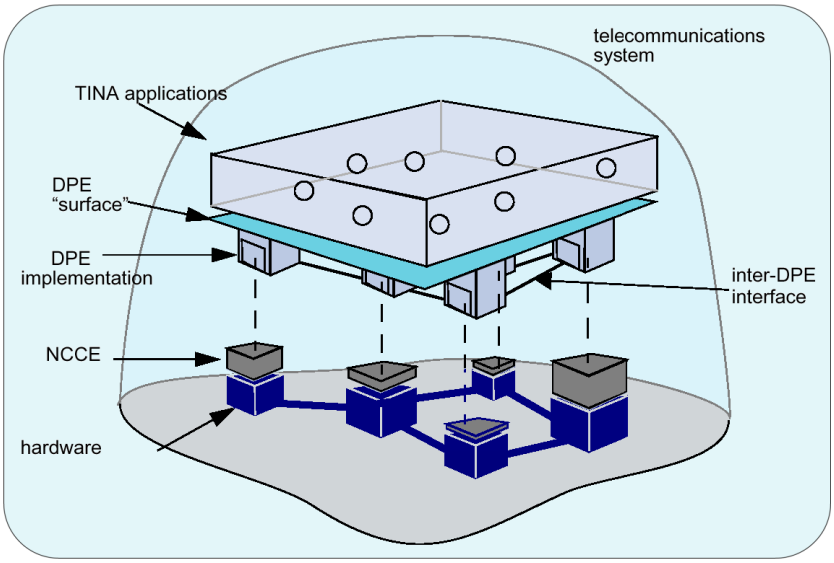


Una applicazione è basata su

- **Servizi**
- **Risorse**
- **Elementi**

Architettura non trasparente

Anche possibile una visione non trasparente con il massimo della visibilità necessaria nel progetto e durante la implementazione



Una applicazione è basata su

- **DPE**
- **Inter-DPE**
- **NCCE**

Progetto di Middleware – Problemi

Il **progetto dei middleware** tende a dovere considerare sempre più alcuni fattori critici che derivano dalla complessità intrinseca della soluzioni

Il primo problema è l'insieme **crescente delle funzionalità** (oggetti, ricorse, ecc.) che introduce un formidabile problema di **scalabilità**

- I middleware tendono ad introdurre **meccanismi indiretti e dinamici (interception)** per la gestione, con un **overhead purtroppo rilevante da minimizzare**

- I middleware tendono a dovere introdurre dei **costi di gestione** che richiedono anche **strumenti sempre più sofisticati e continuamente adeguati** (monitoring, accounting, sicurezza, controllo, ecc.)

- I middleware incorporano **dispositivi mobili e dinamici,** con una necessità di **adattamento continuo alla situazione corrente**

SCENARI di uso di un Middleware

I **middleware** propongono un **modello architetturale** e degli **strumenti**, secondo una **visione di utilizzo precisa**: in questo modo, sono adatti per situazioni di uso molto diverse e implicano un **impiego applicativo** e un **utilizzo raccomandato** da parte di chi lo adotta

Ovviamente possiamo pensare ad usi generalizzati, ma c'è sempre una idea precisa di chi saranno gli utilizzatori e dei loro requisiti

- Un middleware può essere pensato per **una applicazione e che deve funzionare in modo preciso, non flessibile ed isolato:** **requisiti basso costo e bassa intrusione**
- Un middleware può essere pensato per **applicazioni che devono funzionare insieme sul middleware stesso, in modo flessibile:** **requisiti veloce integrazione e comunicazione facilitata**
- Un middleware può **rappresentare la organizzazione** e richiedere una **valutazione continua** dei servizi della stessa con **applicazioni molteplici e servizi adeguati**: **requisito tempo di vita**

MIDDLEWARE a MINIMO COSTO

Un primo scenario è quello di un **middleware** a **basso costo**, che guida la configurazione di **una applicazione**, secondo un **modello di interazione interno**, senza scenari dinamici

Gli utenti richiedono di configurare l'architettura e di ottenere la funzionalità di **una applicazione in modo chiuso e senza variazioni** con servizi di MW a **bassissima intrusione** e **bassissimo costo**

Middleware a scomparsa

Middleware come i **MOM** sono di questa categoria

Si definisce una applicazione che coinvolge un certo numero di nodi e che prevede solo alcuni partecipanti **staticamente determinati** (solo risorse hw e componenti architetturali previsti), con interazione **predefinita**, **rigida** e non variabile e con **costi ottimizzati e bassissimi**

Nessuna necessità di **servizi di supporto** alla **dinamicità**, come un servizio di nomi o altro

Nessun supporto per possibili ingressi o spegnimento di **risorse**

MIDDLEWARE per APPLICAZIONI VELOCI

Un secondo scenario è quello di un **middleware** per **applicazioni molto snelle e ottimizzate**, che richiedono servizi e li ottengono in modo veloce ed efficiente

- Le applicazioni possono fornirsi **servizi reciprocamente**, il **middleware** usa le funzioni proprie e quelle al momento disponibili in modo dinamico
- Supporto per una **gestione dinamica** delle risorse e **applicazioni** che si adattano alla situazione corrente di uso

Middleware per favorire la interazione delle applicazioni

- Middleware Microsoft sono di questo categoria
- Il middleware si **installa anche su richiesta delle applicazioni** che possono **interagire in modi vari** (DOC) con **altre applicazioni attive** al momento della esecuzione
- Supporto alla **dinamicità e anche possibili scelte ottimizzate attuate in modo automatico o su indicazione utente**
- **Tempo di vita del middleware legato alle applicazioni**

MIDDLEWARE per CONTINUITÀ

Un terzo scenario è quello di un **middleware** che deve estendere il **tempo di vita** dei servizi, vedendoli come l'insieme di tutte le funzionalità che una organizzazione può mettere a disposizione per **applicazioni di grana grossa e facilitate**

- Le applicazioni possono anche aggiungere **servizi** che diventano parte del **middleware** e possono essere usati da tutti in **modo anche dinamico**
- Supporto per **gestione dinamica** di risorse e applicazioni

Middleware per ciclo di vita infinito

- Middleware CORBA di questo categoria
- Il middleware è **installato inizialmente** e **viene popolato anche dalle diverse applicazioni** (DOC), **arricchendosi dei servizi** forniti in modo incrementale e senza soluzione di continuità
- Il supporto permette usi di servizi con diversi gradi di **dinamicità** e **possibili scelte adattate in modo automatico**
- **Tempo di vita del middleware massimizzato (no downtime)**

MIDDLEWARE per CLOUD (?)

Una soluzione **CLOUD**, dal punto di vista **utente**, rappresenta la fornitura di uno scenario di **risorse virtualizzate** per ottenere in modo elastico e veloce la risorse necessarie ad ogni fase richiesta dell'utente (1 utente – 1 provider)

Dal punto di vista del provider, è necessario fornire **servizi (-aaS)**, secondo le SLA e seguendo due principi:

- **Efficienza** per dare risposta a tutti gli utenti
- **Efficacia** nell'usare bene le risorse disponibili

In generale ogni provider sfrutta la meglio le **proprie risorse** con un mappaggio delle diverse configurazioni e delle politiche di QoS per il migliore servizio.

Scenari a tendere: **molti a molti**

– Federazione tra **Cloud provider** per scambiarsi servizi e risorse
– Clienti interessati a non avere solo risorse su un provider ma su **più provider** e potere **bilanciare** secondo le proprie **politiche**
– **Cloud come integratore di risorse software (intero stack o Ipaas, Integration Platform as a Service)**

Message-Oriented Middleware

Centrati su uno strumento di basso livello come lo **scambio di messaggi** per mettere insieme parti diverse autonome con obiettivi di **persistenza e di basso costo (**MW come **collante)**

Per un utente:

- Si devono definire in modo **statico mittenti e destinatari** di messaggi
- I messaggi devono essere specificati in **modo preciso**,
  - punto-punto, multicast,
  - formato
- Si possono **mandare** e ricevere **messaggi** (tramite il MW)
  - API tipo *send* e *receive*
  - anche uso di RPC per invocazione

MOM (Glue Middleware)

Dal punto di vista del supporto allo **scambio di messaggi**, i MOM devono rispondere alle API in modo veloce e efficiente

Caratteristiche principali MOM (anche detti **a scomparsa)**

- **Standardizzazione dei messaggi scambiati**
- **Disaccoppiamento temporale e spaziale** di chi comunica senza richiedere **presenza contemporanea** degli interessati
- **Interoperabilità** dei messaggi tra sistemi diversi
- **Permanenza dei messaggi** per operatività anche in caso disconnesso
- **Interconnessioni applicative decise staticamente (!)**

Uso di **specifiche standard** e di **standard di interazione** tra **sistemi diversi** (**API** come **Java Message Service**)

DEPLOYMENT dei MOM

Il **deployment specifico** e il **grafo di interconnessione** è **sempre statico (senza bisogno di sistemi di nomi)**

Modello centralizzato

MOM con sito centrale che funziona hub-and-spoke e si incarica di supportare e smistare tutti i messaggi tra i **clienti diversi**

Modello distribuito

MOM che si decentralizza su tutti i nodi (reti) dei clienti e che prevede di avere degli scambi P2P tra i nodi che sono interessati alla comunicazione

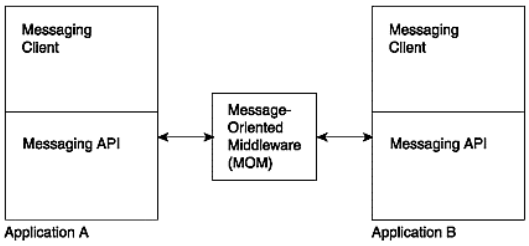
Modello a rete overlay tra applicazioni diverse con precise entità di supporto nel distribuito

- **Necessità di Routing** organizzato ad alto livello
- **Trattamento dei dati** al passaggio tra **ambienti diversi**
- **Entità** partecipanti **predefinite** e **statiche**

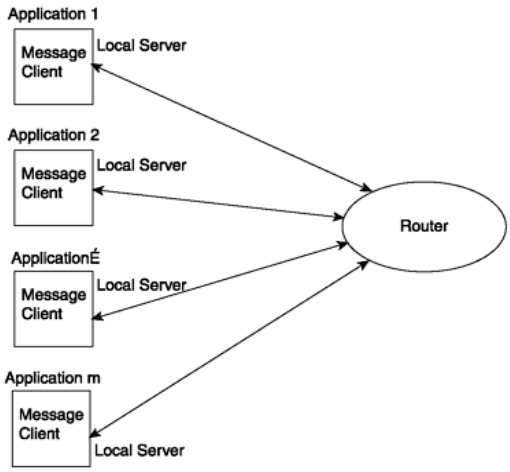
MIDDLEWARE MOM

MOM come fornitori di servizi semplici

operazioni di comunicazione disponibili via API locali ad-hoc



MOM mettono insieme
nodi diversi e forniscono servizi
su molti **nodi di fruizione**
predisponendo **code per il supporto di ogni comunicazione**



MOM come **integratori**

uso di router e di loro interconnessione e di trasformazione di formati

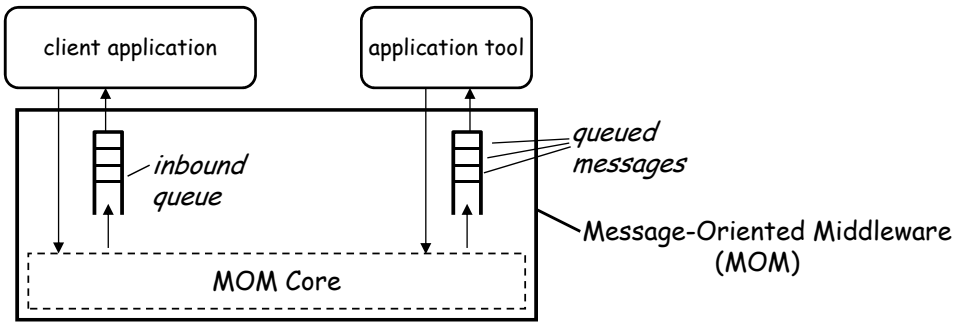
Message-Oriented Middleware

MOM prevedono **code locali ai diversi interessati**

Code sorgenti e **destinatarie** sulle **diverse macchine** interessate (collegate in modo univoco tra di loro)

Gestori delle code garantiscono la corretta operatività e mantenimento ed inoltro dei messaggi

Sistema di routing per i collegamenti delle diverse code
(come una **overlay network** per routing a livello applicativo)



MIDDLEWARE MOM o GLUE

Modello collante o 'glue'

MOM mettono insieme **sistemi diversi** e organizzano la loro **interconnessione**

Relay come entità **intermediarie** per rendere scalabile l'implementazione e organizzare il **routing** di alto livello

Message Broker come entità per supportare il passaggio tra **ambienti** con **rappresentazioni diverse**

Le operazioni dei **MOM** sono via messaggi **non** solo **asincroni punto-a-punto**, ma anche **molti-a-molti**

Il **costo** delle **realizzazioni** deve essere **limitato**:
si tende alla rapida integrazione di **sistemi legacy esistenti**

MOM: MQSeries IBM

Proposta di MOM molto diffusa e supportata

Tipicamente, il **grafo di interconnessione** (**routing**) è sotto il controllo di una gestione di sistema **sempre statica** e **poco flessibile (nessun sistema di nomi)**

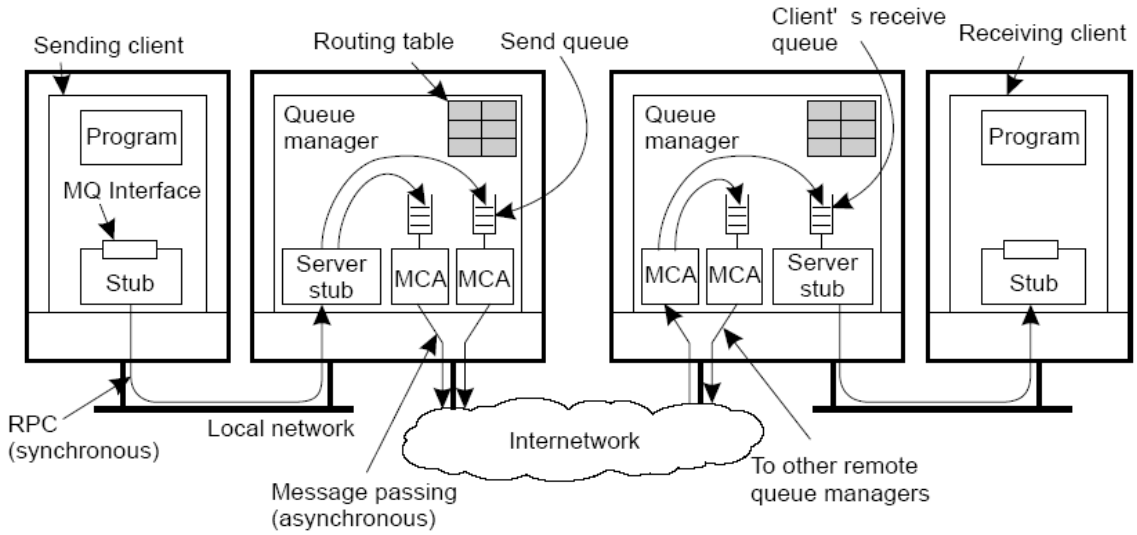
Messaggi applicativi sono gestiti da **queue manager**
I processi interagiscono con *API RPC* per mettere /estrarre messaggi dalle code locali

Il trasferimento avviene attraverso **channel** unidirezionali gestiti da **Message Channel Agent** che si occupano di tutti i dettagli (politiche diverse di consegna, tipo di messaggi, ecc.)

Il **coordinamento tra MCA** viene attuato attraverso **primitive** che dovrebbero facilitare il coordinamento (politiche diverse di attivazione, durata, costo massimo, mantenimento dello stato, ecc.)

MQSeries IBM – parte di Websphere

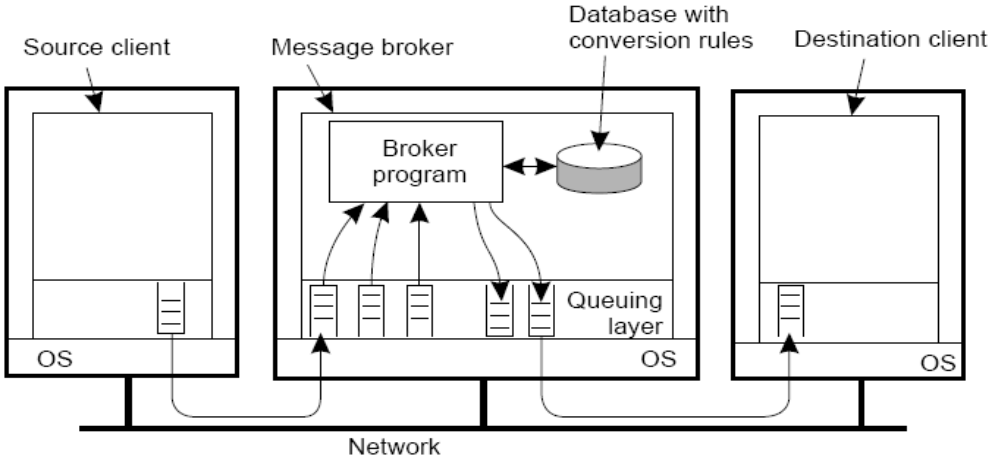
Per il deployment, **l'amministratore** stabilisce le **opportune interconnessioni** con tabelle di **routing** alla **configurazione alla inizializzazione**



MQSeries IBM: Broker

Per favorire la integrazione, **MQ Broker** possono intervenire sui messaggi

- trasformando **formati**
- organizzando **routing** in base al **contenuto**;
- lavorando su **informazioni di applicazione**, per **specificare sequenze** di azioni



MIDDLEWARE OO e DOC

SISTEMI APERTI e DISTRIBUITI ad OGGETTI

modelli ad oggetti (componenti) per la **integrazione e interazione reciproca completa tra oggetti presenti contemporaneamente in ambienti eterogenei (middleware del secondo e terzo tipo)**

Middleware intrinsecamente più complessi e corposi in cui ogni servizio deve essere disponibile sempre per ogni possibile richiedente

- definizione di una **interfaccia** per oggetto
- **interazione** tra **oggetti** attraverso i **riferimenti**
- **bus** per **integrazione** di oggetti **progettati in linguaggi diversi**
- definizione della interazione anche tra sistemi diversi con diversi gestori

Oggetti con supporto al **diverso tempo di vita**

MIDDLEWARE OO basati su C/S

ENTITÀ da mettere in relazione: ***cliente*** e ***implementazione*** di un ***oggetto*** *per il* ***servizio***

Binding sempre dinamico tra clienti e server attraverso **riferimenti dinamici**

oggetto	entità che fornisce servizi
riferimento	accesso al servizio
tipo	entità per classificare gli oggetti
interfaccia	descrizione delle operazioni per un insieme di oggetti
richiesta	meccanismo per manifestare esigenza di un servizio
operazione	servizio con nome che può essere richiesto ad un oggetto

Diverse implementazioni:

Entità centralizzate ⇨ **CORBA e bus ORB**

Modello P2P ⇨ **DCOM e connessioni dirette per ogni cliente e servitore**

CONCETTI COMUNI ai MIDDLEWARE

Una serie di pattern tipici descrivono la interazione in DOC Middleware [Volter2002]: Pattern per

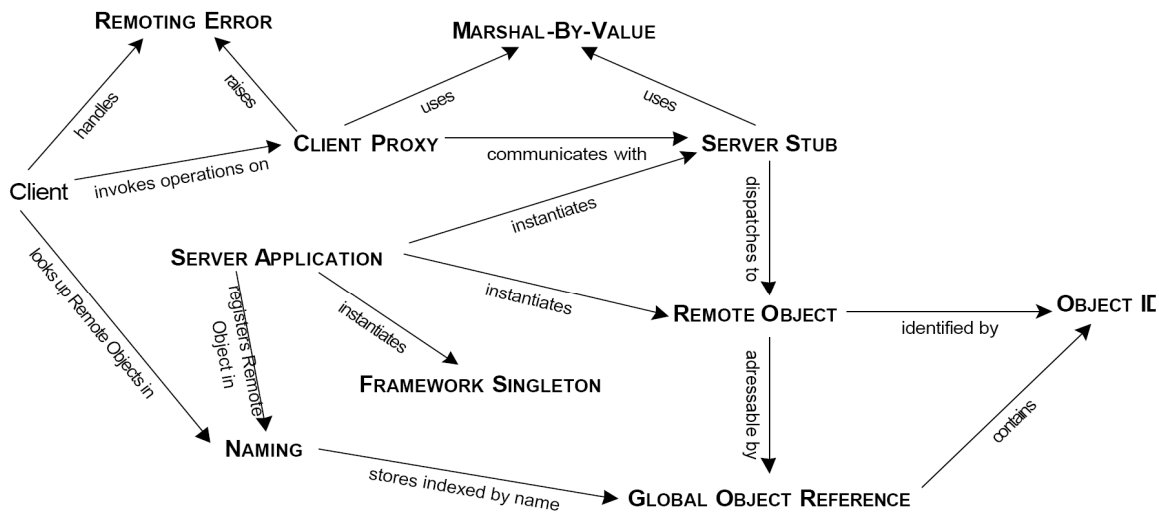
- **Interazione remota: Basic remoting**
- **Comunicazione asincrona: Asynchronous Communication**
- **Gestione risorse: Resource and service management**
- **Ulteriori servizi: Additional Services**
- **Qualità del servizio: Quality of Service**

Ci permettono di ritrovare concetti e modi già visti e anche presenti in CORBA e in altri DOC

Basic remoting Pattern

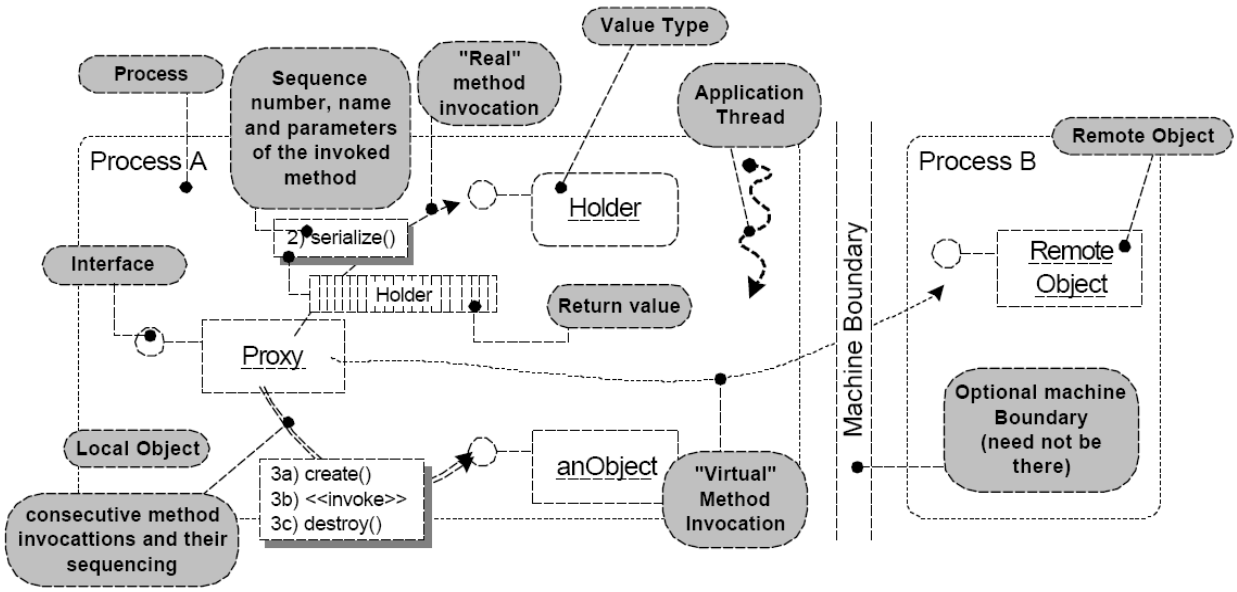
Basic remoting Pattern

Relazioni per cominciare a considerare oggetti remoti



Legenda

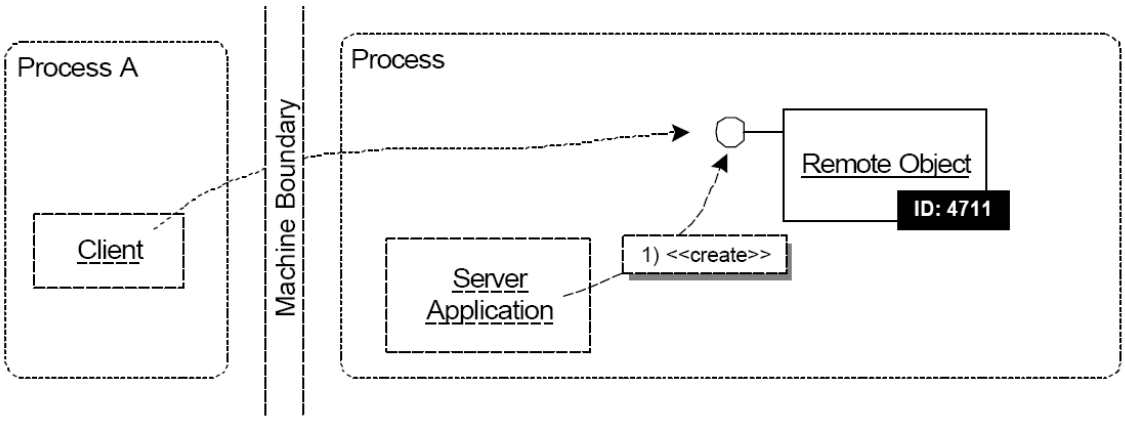
I diagrammi non vogliono essere formali ma solo indicativi per dare una idea delle interazioni



Basic remoting Pattern: Remote Object

Remote Object: riferimento remoto per oggetto remoto

Per avere la possibilità di **riferire un oggetto non locale già istanziato e da raggiungere** su un sistema remoto

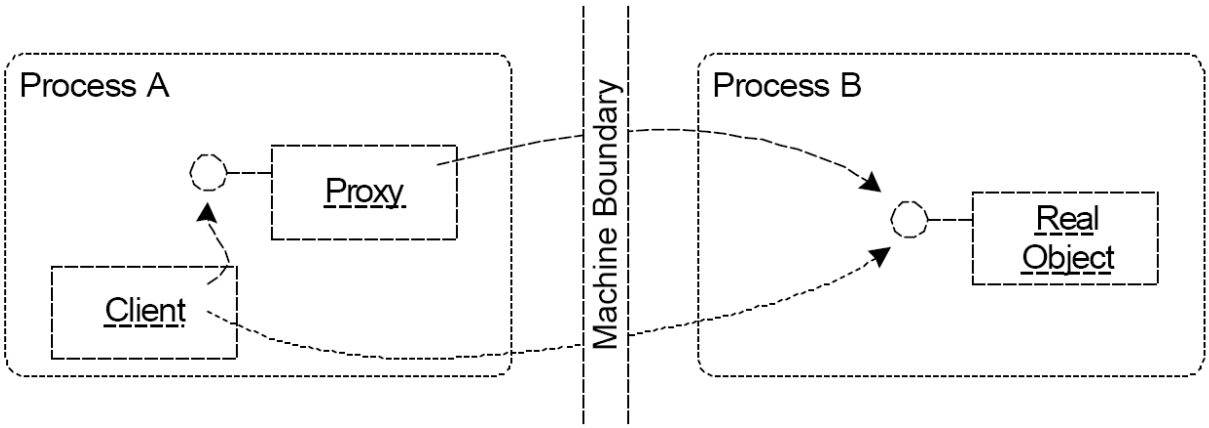


Creazione anche fatta in remoto e poi riferimento disponibile

Basic remoting Pattern: Proxy

Proxy Lato Client

Intermediario locale al cliente per arrivare all'oggetto **server Remoto**

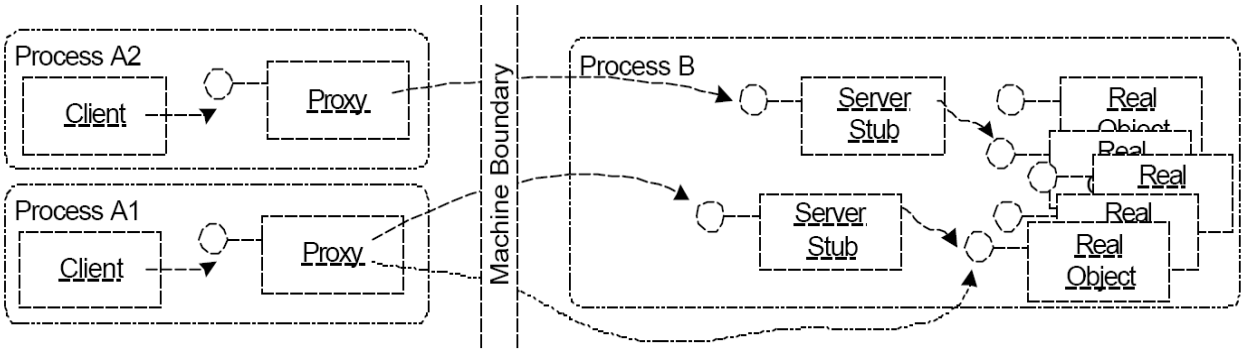


Proxy permettono di accedere ai server come se fossero locali

Basic remoting Pattern: Stub

Stub Lato Server

Intermediario visibile da remoto per arrivare all'oggetto locale Server

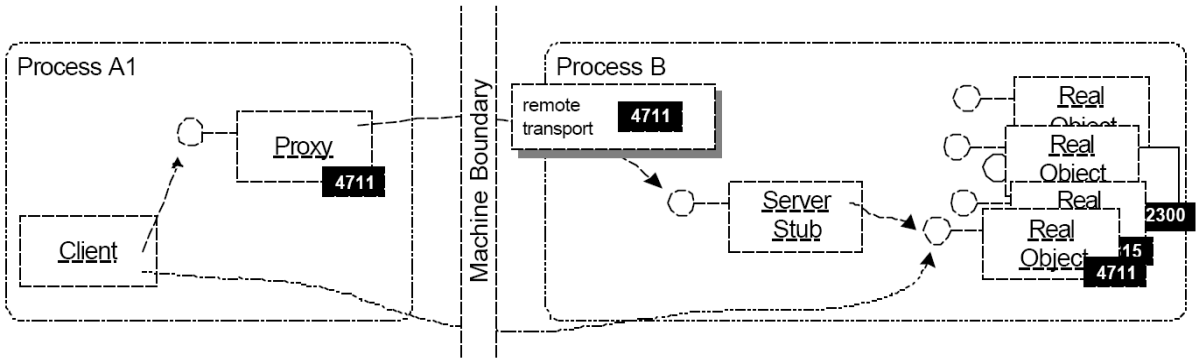


Stub sono acceduti da remoto dai **Proxy** per arrivare ai server (coordinamento proxy e stub)

Basic remoting Pattern: ObjectID

ObjectID – Identificazione unica di oggetti remoti

Per avere la possibilità di riferire un oggetto specifico da remoto

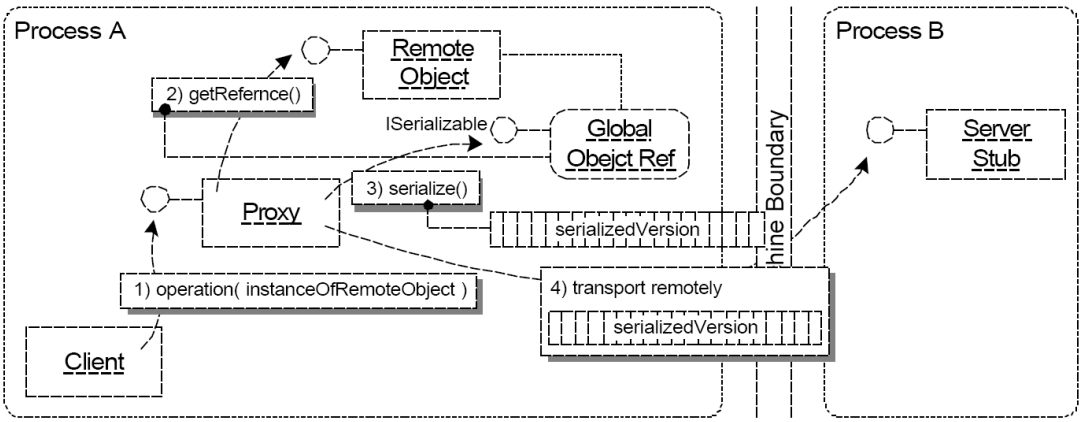


Ogni oggetto remoto denotato da un identificatore globale unico, senza possibilità di ambiguità (ad esempio, usando la coppia: *nome host* + *nome locale*)

Basic remoting Pattern: ObjRef

Global Object Reference – Nomi Globali Unici

I riferimenti globali ad oggetti permettono di passare conoscenza di oggetti remoti (fissi e non mobili) ad altri oggetti

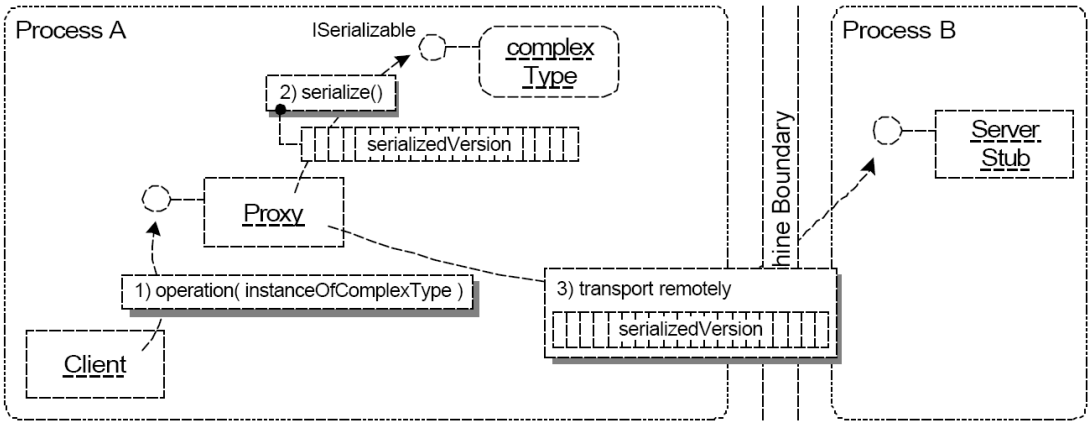


Si permette ai clienti di passare indicazioni di oggetti da potere raggiungere in remoto, e anche attributi addizionali

Basic remoting Pattern: Obj ByValue

Passing by Value (Marshal-by-value)

Per passare argomenti per valore che il server possa usare al bisogno

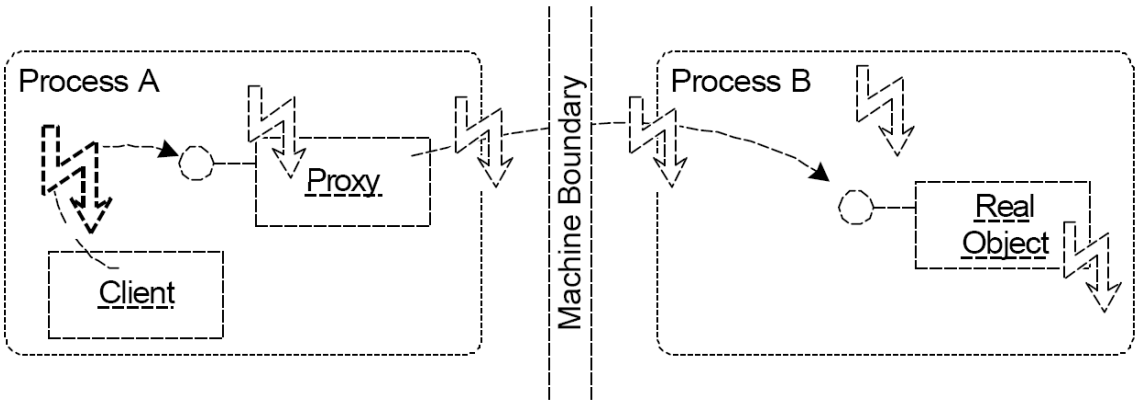


I valori sono passati per copia, in genere, e serializzati

Basic remoting Pattern: RemoteError

Remoting Error (Passaggio di errore)

Una operazione può causare anche errori che devono essere gestiti al meglio (da tutte le entità interessate)

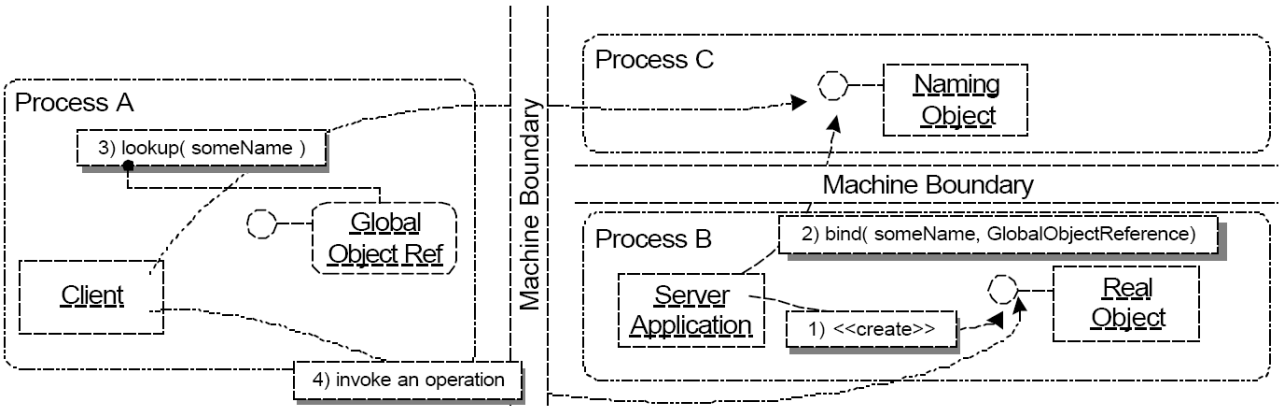


La gestione può dovere essere fatta sia dal server sia dal client

Basic remoting Pattern: Naming

Naming Support (Supporto di nome)

Un riferimento remoto può essere ottenuto con il supporto di un sistema di nomi

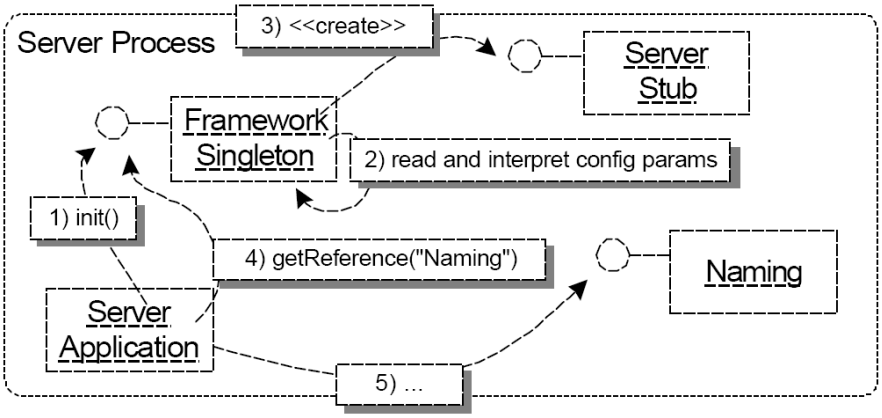


Il cliente ottiene il riferimento remoto da un sistema di nomi noto che deve essere disponibile su richiesta

Basic remoting Pattern: Singleton

Framework Singleton (Oggetti Unici del Framework)

Molti componenti devono essere configurati e inizializzati correttamente in modo complesso una volta per tutte



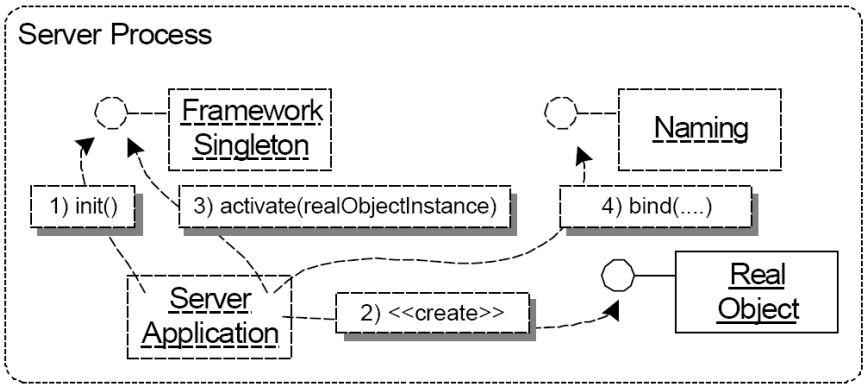
Il cliente delega un oggetto singleton come interfaccia (facade) per le operazioni iniziali

Basic remoting Pattern: ServAppl

Server Application

(Attivazione e Deployment di oggetti remoti)

I server devono essere attivati secondo strategie diverse e adatte alle diverse esigenze

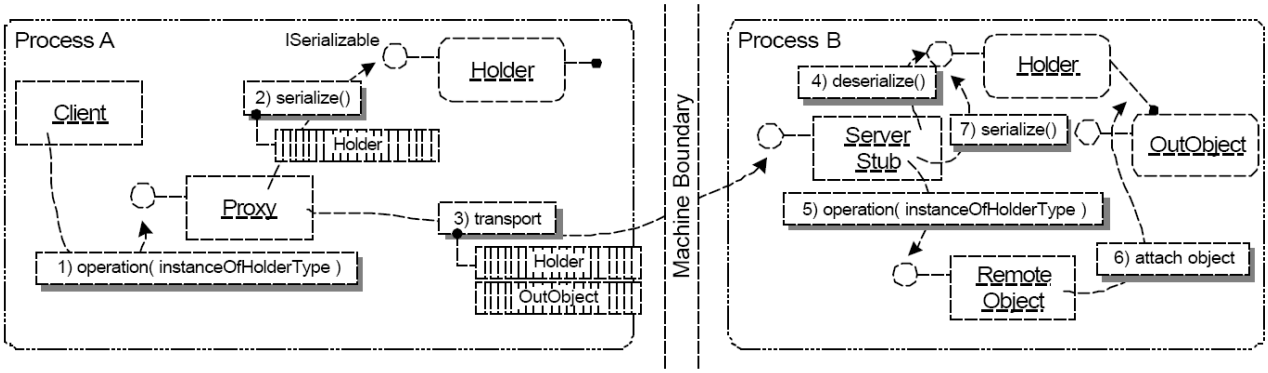


Il cliente può anche intervenire sulla politica di attivazione della infrastruttura remota richiedendo politiche specifiche

Basic remoting Pattern: Holder

Holder (Adattamento semantico)

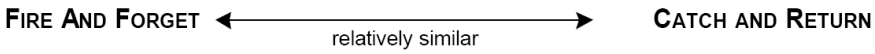
In alcuni casi, bisogna prevedere degli adattatori per consentire linguaggi che non prevedono parametri di out e altri che li prevedono



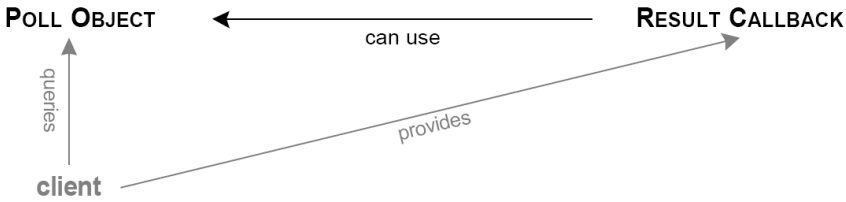
Il cliente usa un **holder** che viene passato per contenere l'oggetto in ingresso e da cui recuperarlo in uscita per poterlo usare (*read e write come operazioni*)

Asynchronous Communication Pattern

La **invocazione** può anche **non** essere **sincrona ma**
Asincrona - nessun risultato
Sincrona non bloccante - risultato recuperato poi



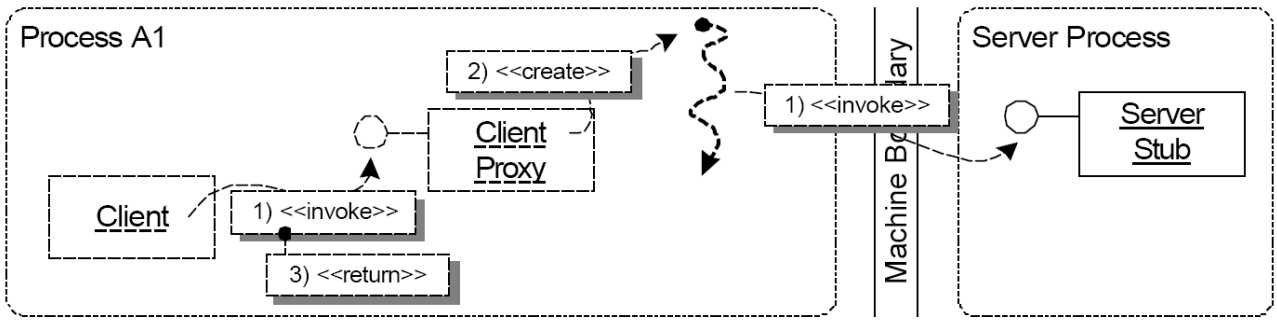
Fire & Forget
Catch & Return
Poll Object
Result CallBack



Asynchronous Communication Pattern

Fire & Forget / Spara e Dimentica

Per semplici operazioni (Asincrone) in cui non ci sia (interesse per) il risultato



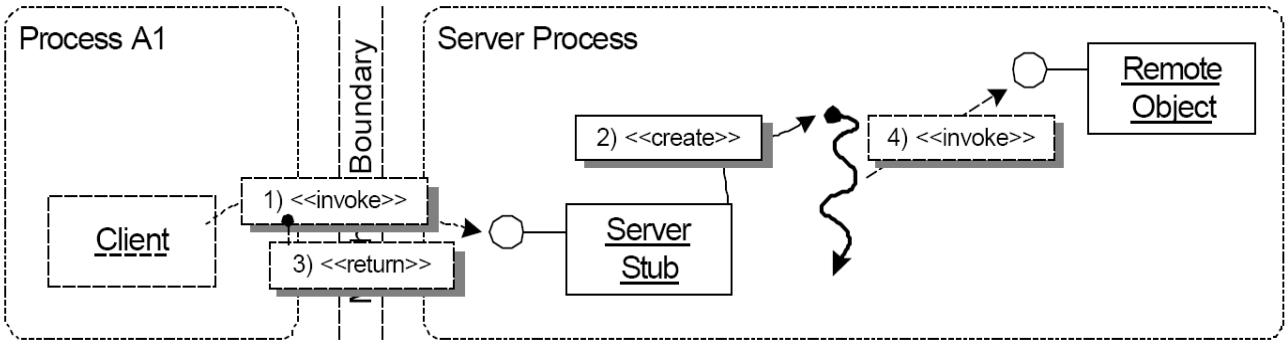
Il cliente ottiene una semantica *best effort* e non ha notizia di eventuali fallimenti sul server (*operazione asincrona eventualmente bloccante*)

Si genera un thread dalla parte del cliente

Asynchronous Communication Pattern

Catch & Return Invia e Restituisci il controllo

Per operazioni asincrone in cui non ci sia (interesse per) il risultato applicativo ma maggiore sicurezza di consegna

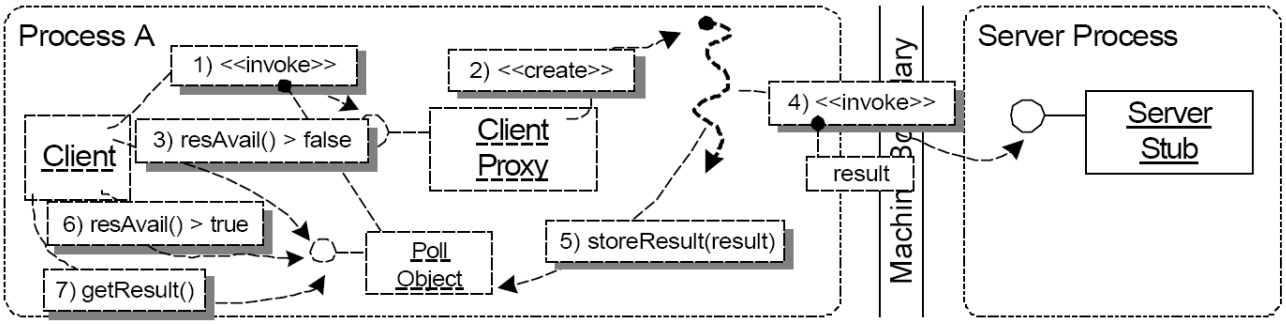


Il cliente ha una semantica più garantita con eventuale notizia di fallimenti sul server (*operazione sincrona a livello di supporto*)
Il server stub genera un thread che si occupa del risultato

Asynchronous Communication Pattern

Poll Object Polling dell'oggetto remoto

Il Cliente ottiene il controllo immediatamente (operazione non bloccante) e una indicazione di un oggetto locale

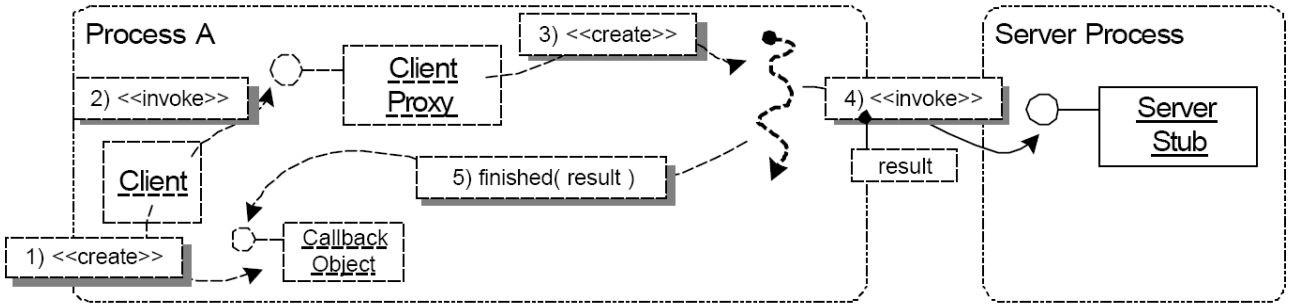


Il cliente deve interagire con il poll object locale: decide se e quando ritrovare il risultato agendo localmente sull'oggetto poll

Asynchronous Communication Pattern

CallBack Chiamata sul risultato di ritorno

Il Cliente non aspetta ma specifica un handle (callback) che viene invocato al momento del completamento

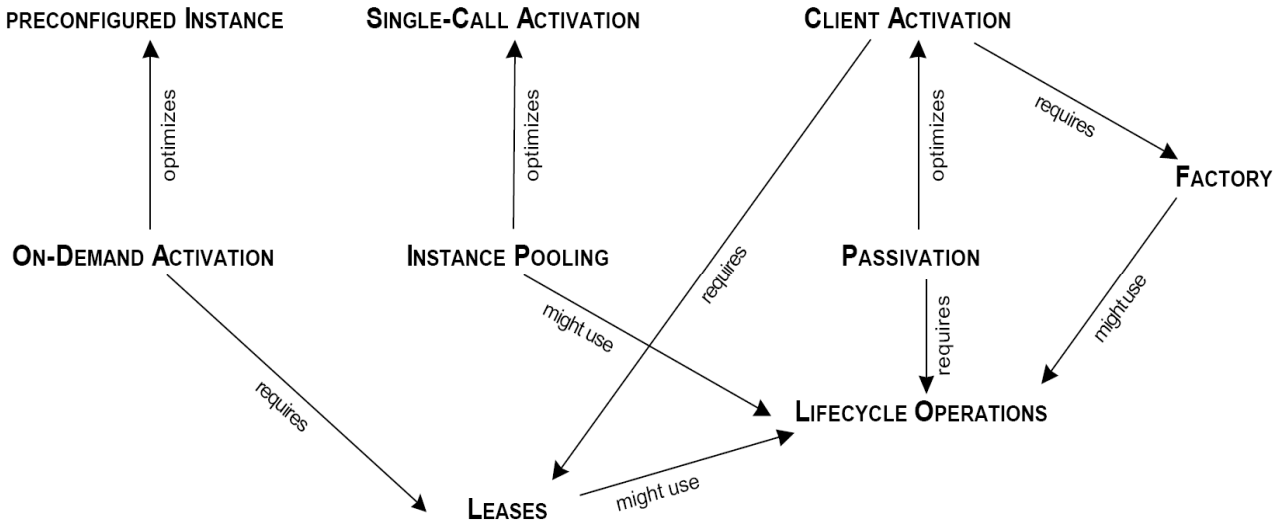


Il risultato viene fornito all'oggetto Callback che ha la responsabilità di trattarloo (attraverso *codice da eseguire invocato automaticamente*)
Il richiedente può disinteressarsene

Resource and service management

Resource and service management

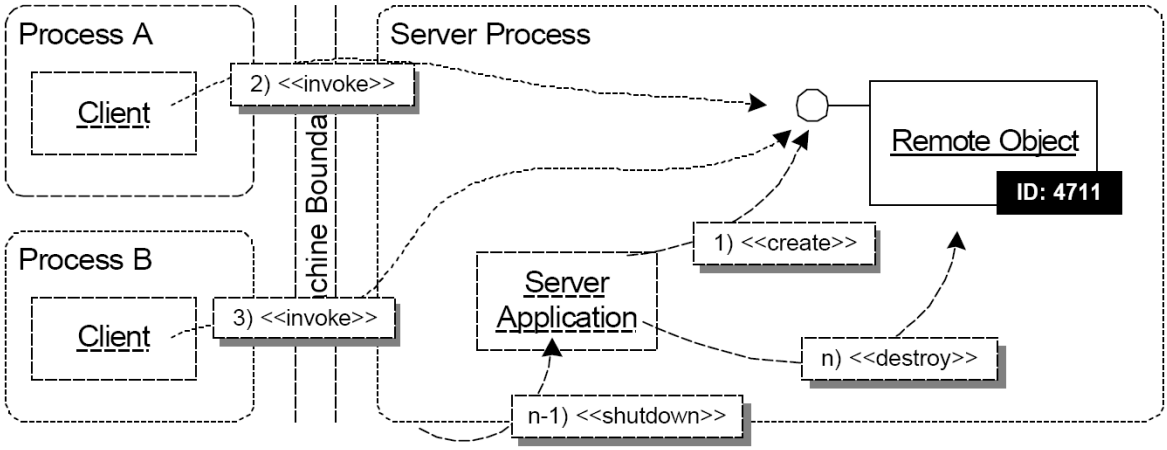
Ci permettono di interpretare tutto il ciclo di vita degli oggetti invocati e di gestire politiche diverse (stato)



RSManagement: Preconfigurazione

Preconfigured Instances – Istanze solo Preconfigurate

Server predeterminato e operazioni fornite a tutti i clienti allo stesso modo (stato?)

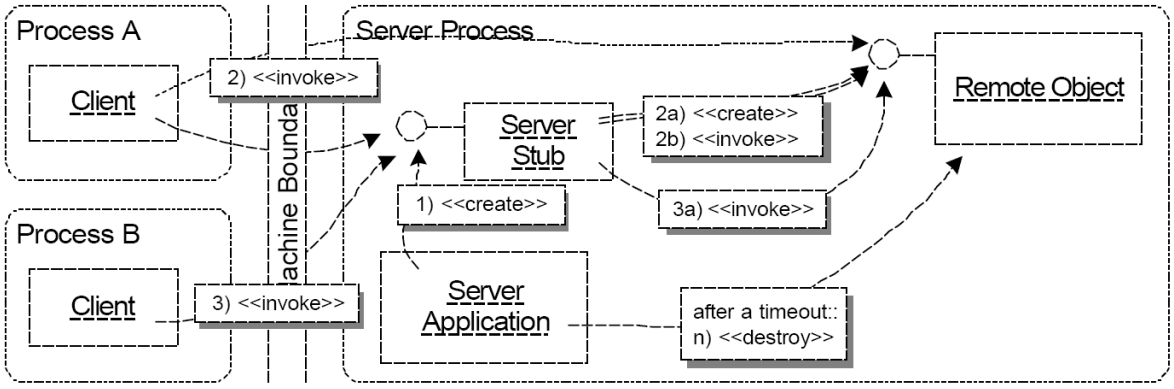


Il servitore opera tra la create e la destroy

RSManagement: Attivazione su Richiesta

On-Demand Activation – Attivazione su Richiesta

Server con stato e operazioni differenziate per clienti diversi. Attivazione solo su richiesta e bisogno

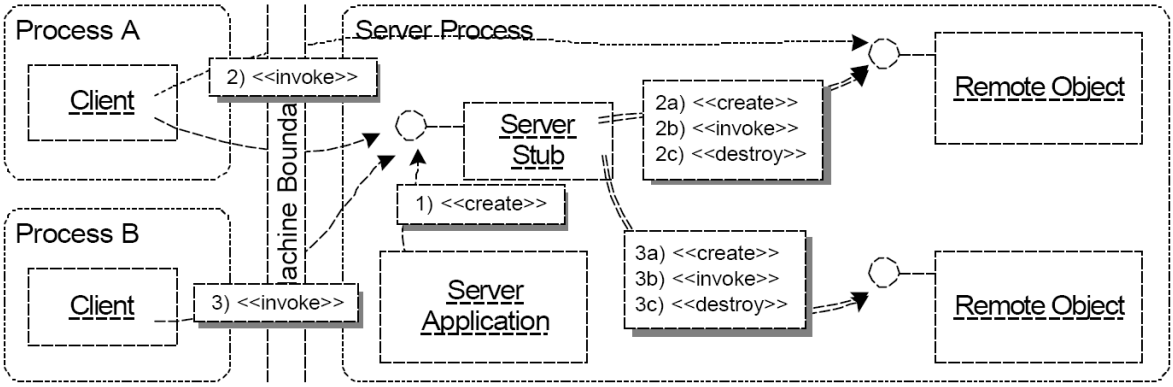


Il servitore opera tra la create e la destroy, che diventa necessaria e deve essere attuata al bisogno

RSManagement: Attiv. ad ogni Chiamata

Single-Call Activation - Attivazione ad ogni richiesta

La istanza viene attivata ogni volta ex-novo, senza problemi o vincoli (si trascura la concorrenza eventuale e si assume che non ci sia stato)

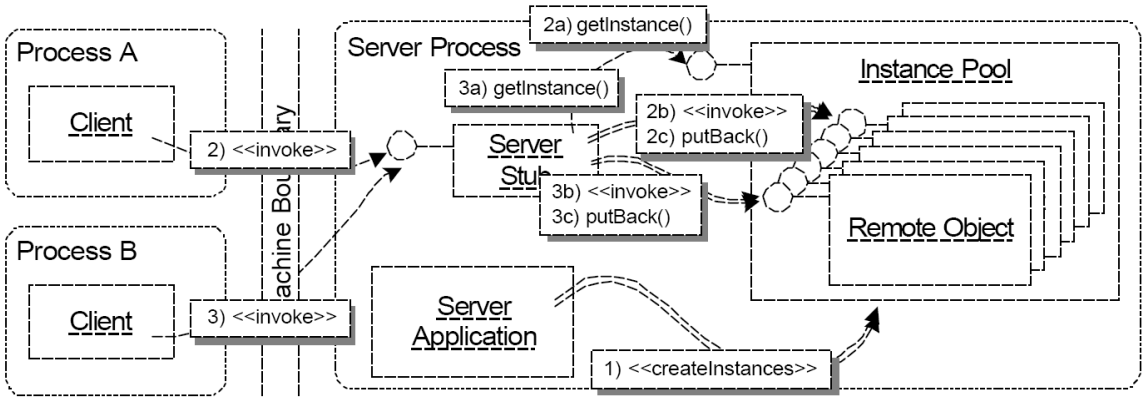


Controllo sul numero massimo di istanze contemporanee
L'overhead di attivazione/distruzione aumenta

RSManagement: Pool di istanze

Instance Pooling - Creazione di un pool di oggetti

Si creano un fissato numero di istanze di oggetto che rispondono ciascuna ad una richiesta (anche variabile con il traffico) con la stessa interfaccia

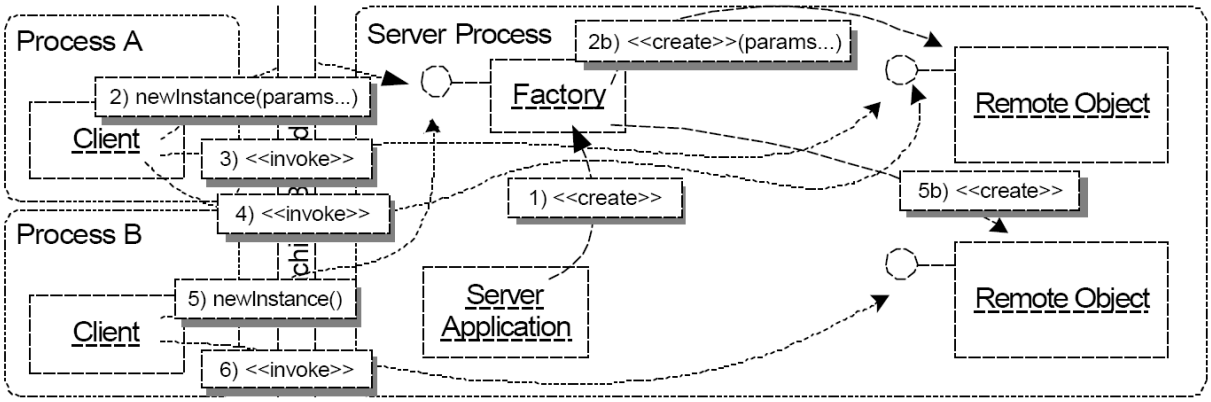


Controllo sul numero massimo di istanze contemporanee per ottimizzare le risorse ed il loro uso

RSMngmt: Attivazione da parte del Cliente

Client Activation - Oggetti su responsabilità del Cliente

Il cliente decide se e quando attivare una istanza del servitore

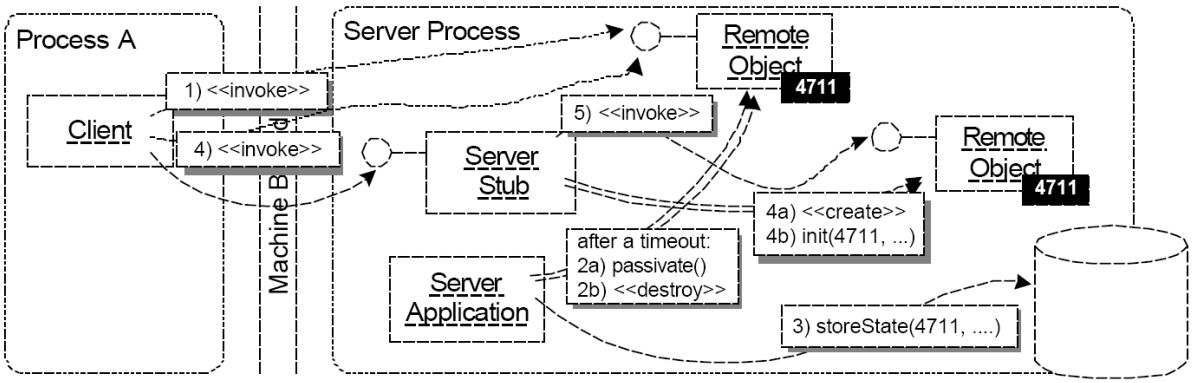


Il cliente mantiene il server (con stato del cliente) come sua responsabilità e deve controllare e gestire la istanza

RSManagement: Passivation

Passivation - Controllo su oggetti

Il servitore può decidere di rendere passivi oggetti gestiti dal cliente

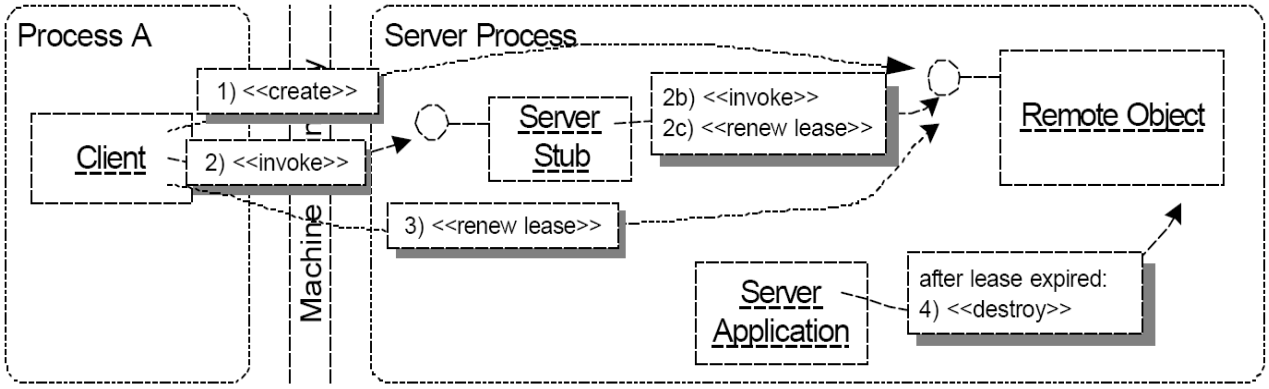


Il servitore limita l'impiego di risorse dalla sua parte rendendo passive le istanze stesse se e quando non necessarie

RSManagement: Lease

Lease - Eliminazione oggetti dopo un certo tempo

Il servitore decide se distruggere istanze non usate

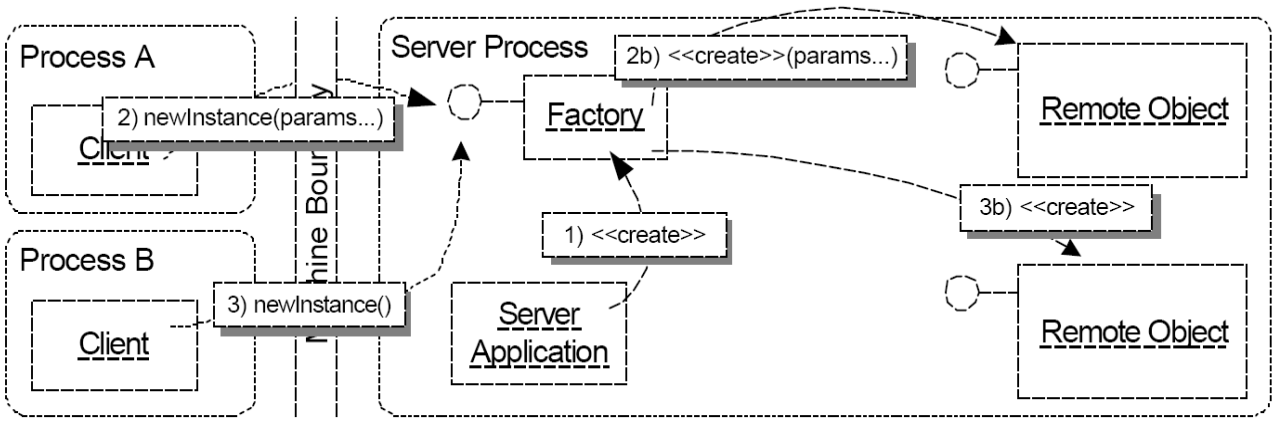


Per fare fronte anche a guasti del cliente, il server decide di eliminare alcune istanze non usate

RSManagement: Factory

Factory - Attivazione di istanze su nodi remoti

Il cliente può ottenere riferimenti ad istanze solo attraverso una Factory (entità deputata a crearne altre)

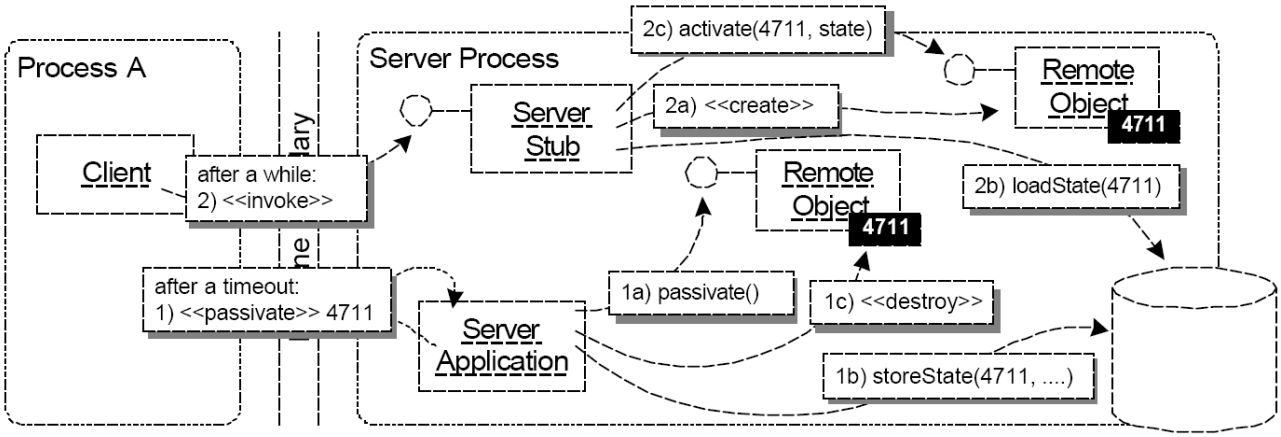


La Factory risiede sul nodo del servizio e attiva (crea) istanze anche per oggetti remoti (factory e non classe)

RSManagement: Gestione lifetime

Operazioni di gestione esplicita del tempo di vita

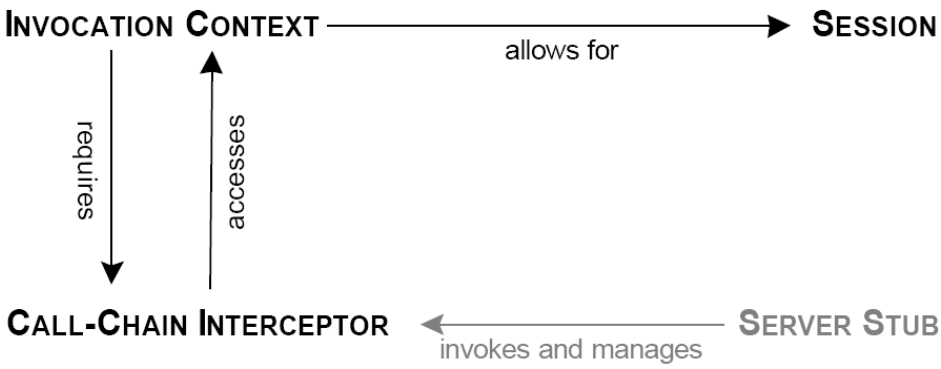
Le istanze stesse possono eseguire operazioni più complesse in relazione al loro tempo di vita



Il server può invocare le istanze per eventi specifici in relazione al tempo di vita delle stesse

Additional Services

Una serie di pattern permettono di fornire servizi addizionali, in dipendenza dal contesto della invocazione, attivati da un evento, che legano le invocazioni



Session

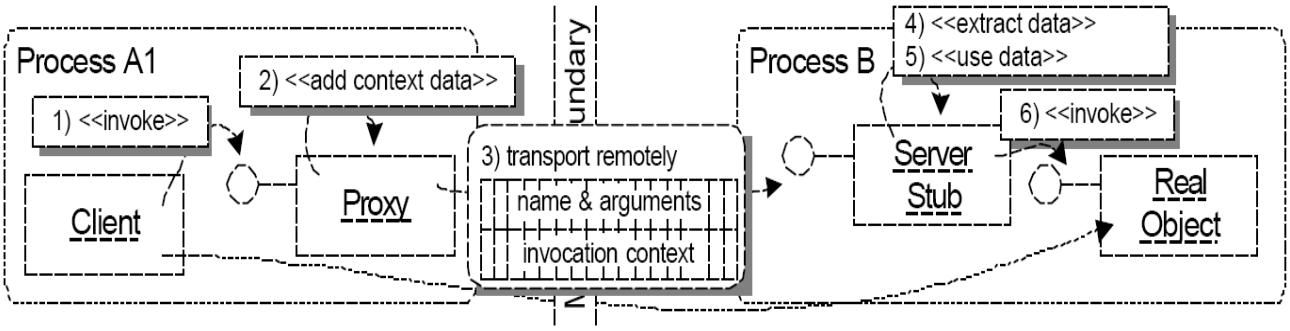
Invocation Context

Call-Chain Interceptor

Additional Services: Invocation Context

Invocation Context (Contesto per le invocazioni)

Le operazioni possono richiedere altre informazioni oltre ai parametri di invocazione (e.g., identificatore di transazione)

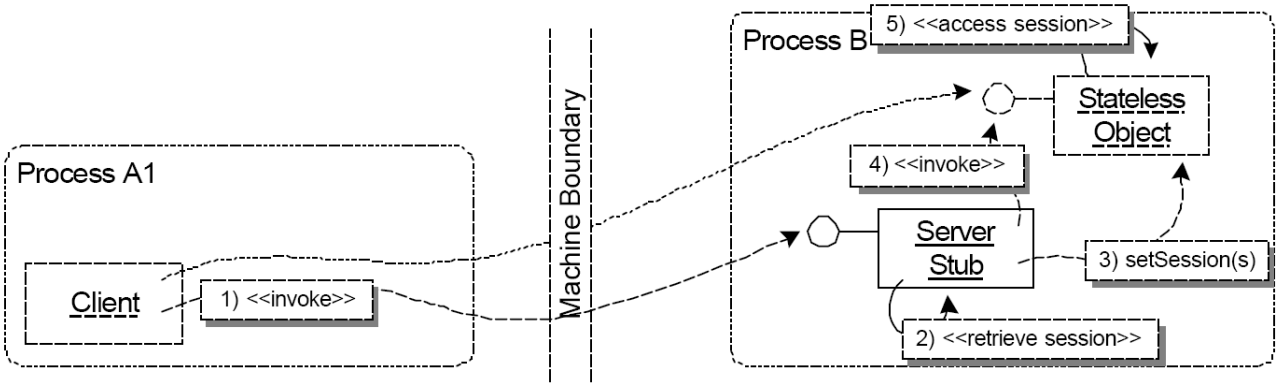


Il contesto permette di ottenere automaticamente alcune informazioni necessarie per lo svolgimento delle operazioni

Additional Services: Session

Session (sessioni sul server per i clienti)

Le operazioni possono richiedere informazioni di stato sul servitore (stato della sessione)

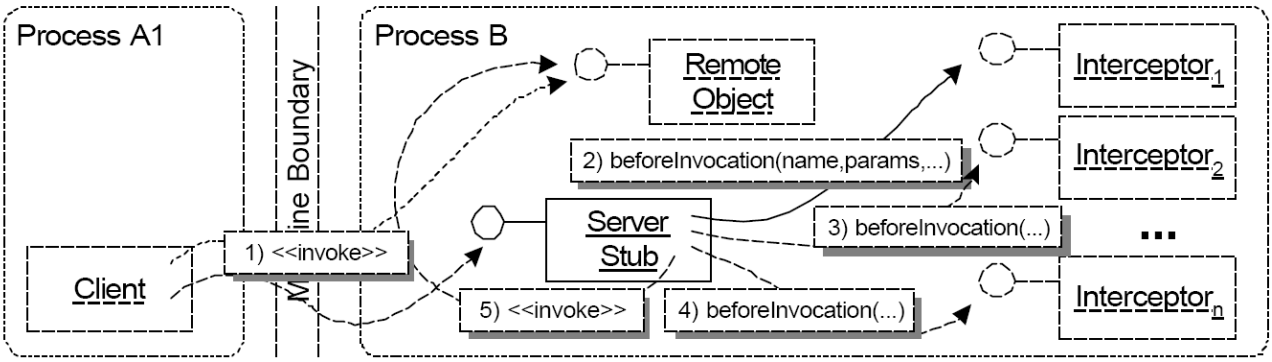


La sessione viene mantenuta sul server per conto dei clienti che fanno operazioni successive
I clienti possono anche avere un supporto leggero

Additional Srv: Call-Chain Interception

Call-Chain Interceptor (componenti sul server per rispondere ad esigenze clienti)

Le operazioni possono richiedere azioni addizionali sul servitore (sicurezza, transazioni, …)



Gli interceptor sono componenti che possono inserire azioni sulla operazione richiesta dai clienti (non solo sul server)

QoS Services

Una serie di pattern permettono di fornire servitori con Qualità differenziate e con migliori performance

Broker

Lifecycle Manager

Custom Marshaller

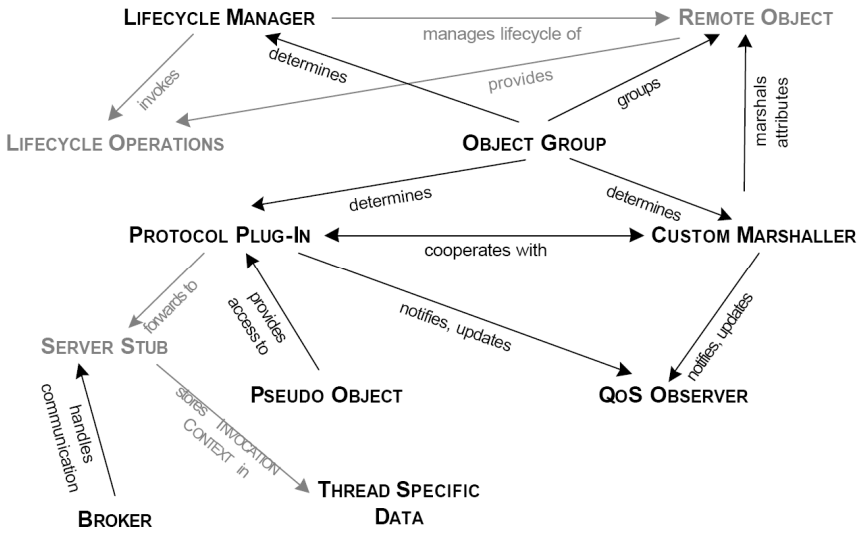
Object Group

Protocol Plug-In

Pseudo Object

QoS Observer

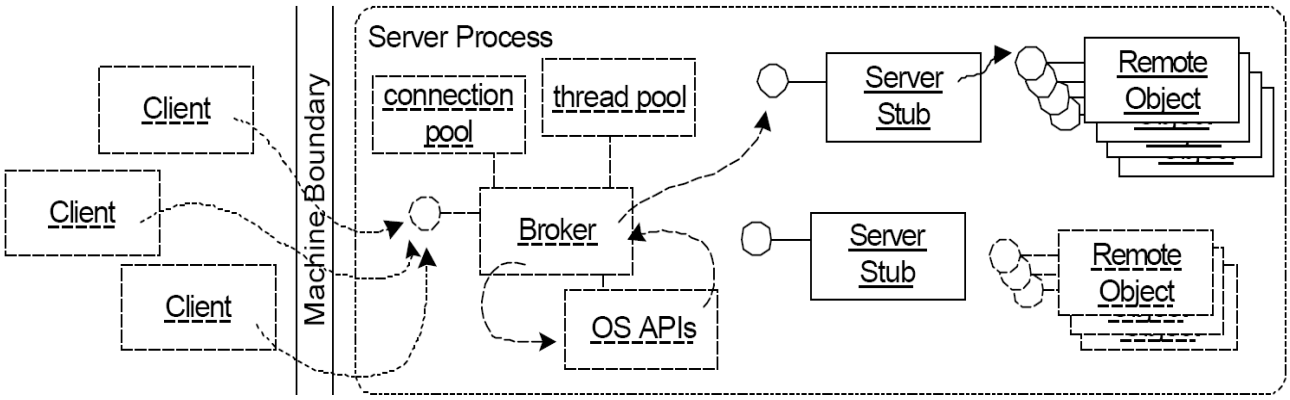
Thread-specific Data



QoS Services: Broker

Broker (Coordinatore delle politiche di comunicazione)

Le operazioni sono facilitate dalla presenza di un gestore centralizzato che coordina

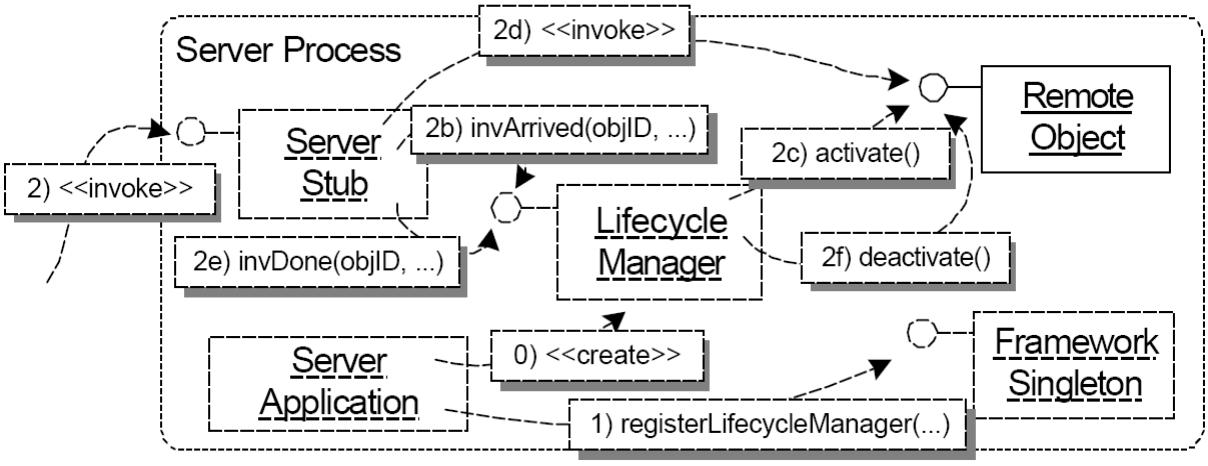


Il **Broker** decide politiche di attivazione e di esecuzione delle operazioni (*vedi **OA** in CORBA*)

QoS Services: Lifecycle manager

Manager del tempo di vita con politiche custom

Alcune politiche di tempo di vita possono essere personalizzate in base a molti parametri

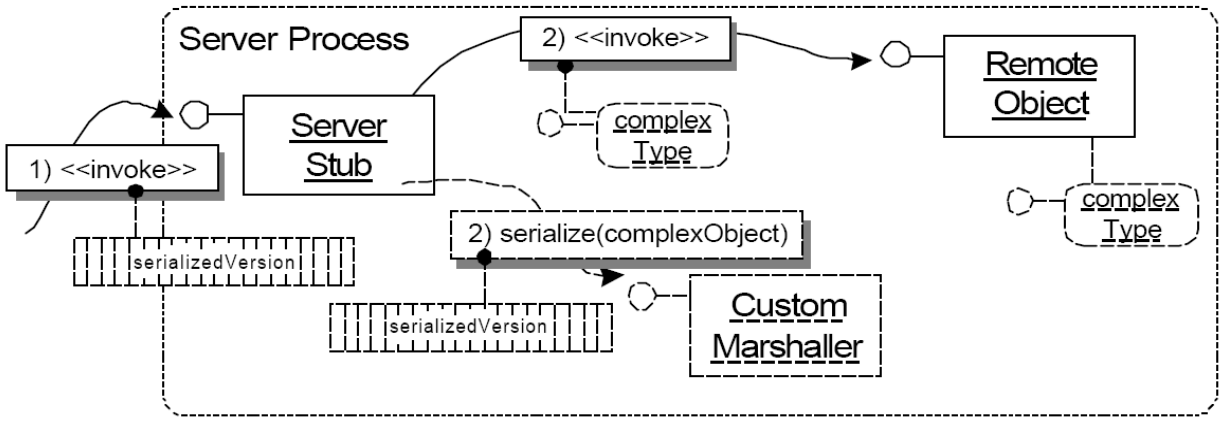


Il **Manager** può introdurre appunto politiche per alcuni degli oggetti in modo differenziato (*vedi **OA** in CORBA*)

QoS Services: Custom Marshaller

Gestione ad-hoc di alcune operazioni di marshalling

Alcune gestioni sofisticate devono essere adattate a secondo dell'ambiente di uso

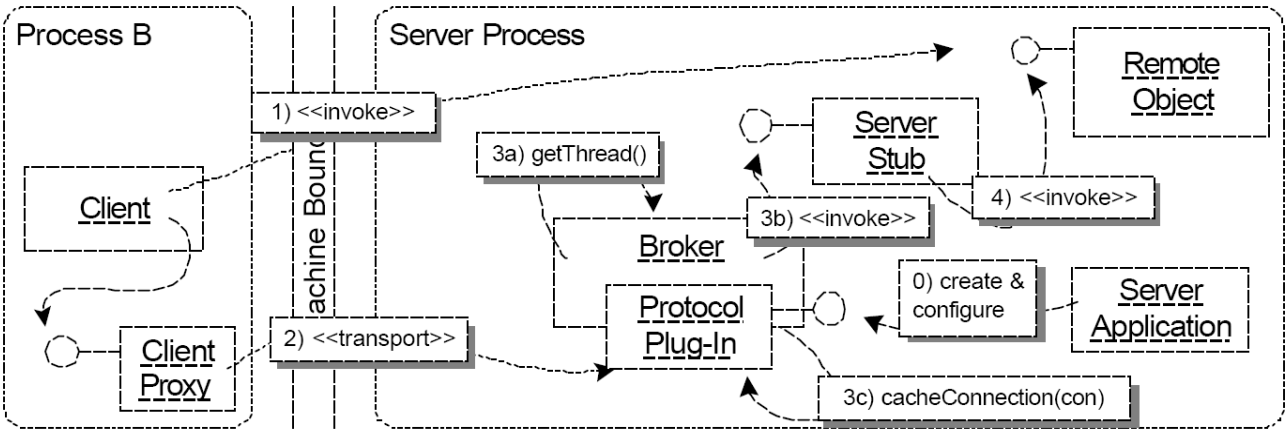


Alcune politiche di marshalling dipendono da molti fattori e possono variare

QoS Services: Plug-in

Protocol Plug-In (Azioni inserite)

Le operazioni sono facilitate dalla presenza di plug-in che possono richiedere protocolli ad-hoc

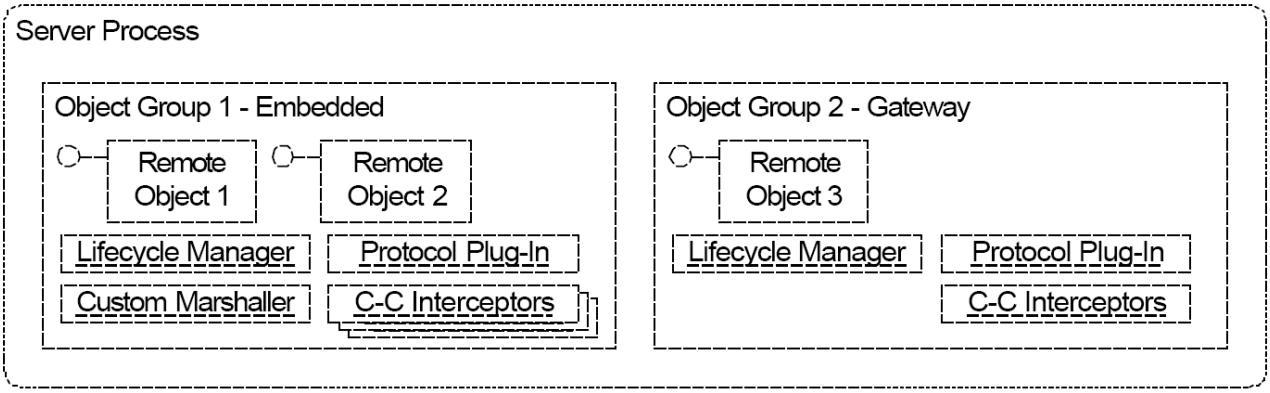


I plug-in consentono di adattare la operazione dalla parte del server tenendo conto della architettura corrente

QoS Services: Object Group

Object Group (Gruppi di oggetti con QoS)

Le operazioni su diversi nodi possono avere proprietà differenziate per QoS diversi

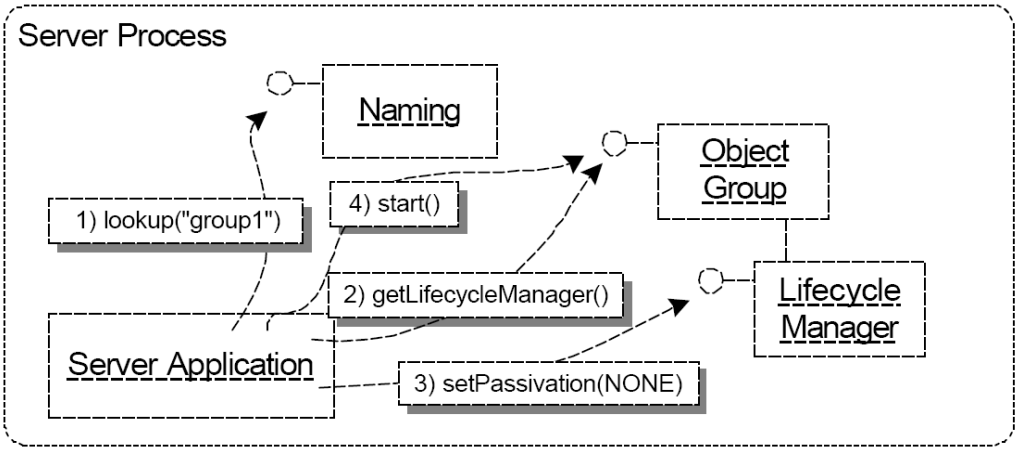


Una applicazione può mettere a disposizione molti gruppi di oggetti per diversi QoS

QoS Services: Pseudo Object

Pseudo Object (Oggetti finti ma visibili)

Alcune operazioni devono avere visibilità di alcuni oggetti senza che si possano modificare da remoto

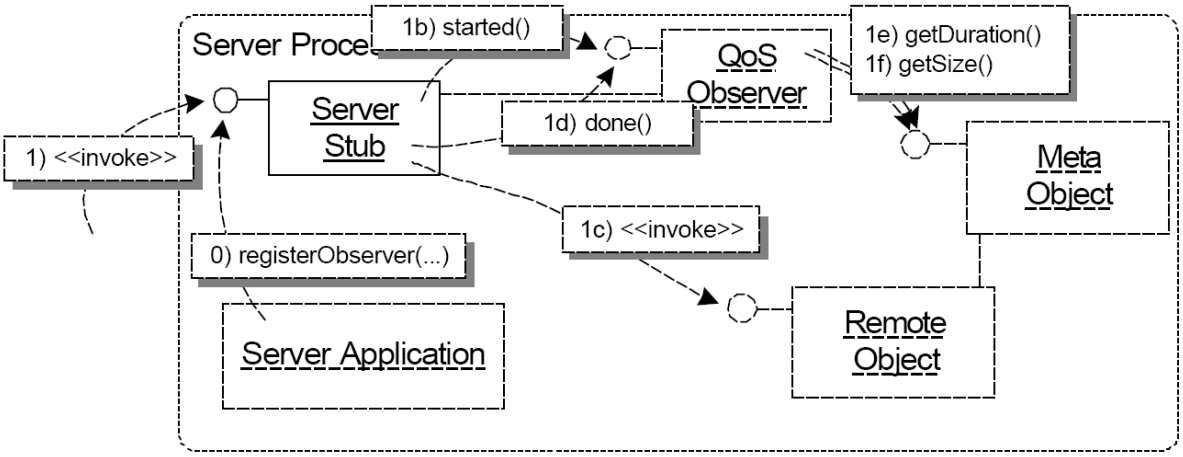


Oggetti che hanno una interfaccia ma non sono tipicamente visibili dall'esterno ma solo dall'interno per comodità di gestione (CORBA pseudo-oggetti)

QoS Services: QoS Observer

QoS Observer (informazioni sulla QoS)

Le operazioni possono anche non mantenere il QoS negoziato prima della esecuzione

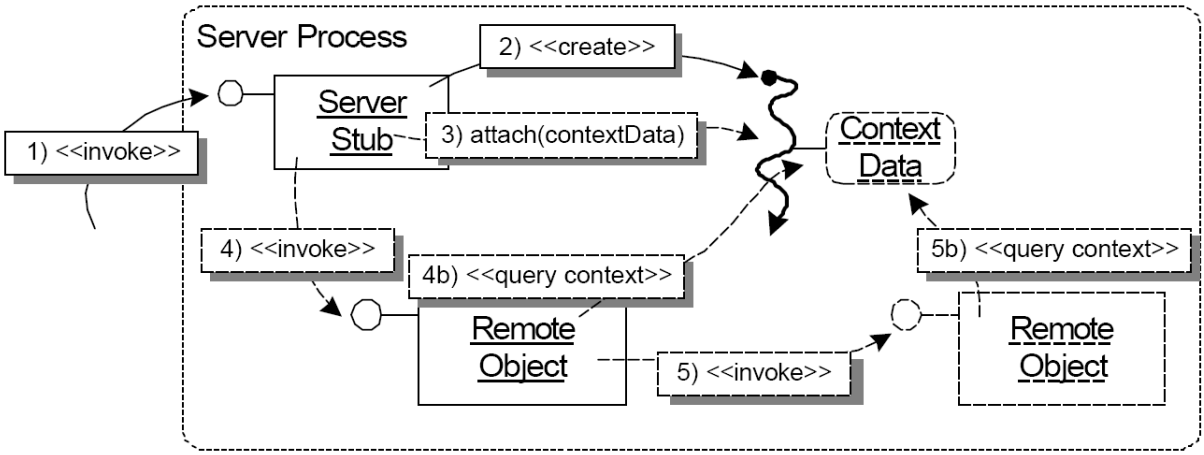


Si forniscono informazioni sulle azioni e sugli eventi causati dalla operazione dalla parte del servitore

QoS Services: Thread-specific Data

Thread-specific data (visibilità di dati addizionali a thread)

Alcuni processi che sono invocati sul server devono avere accesso ad informazioni di contesto



Si forniscono informazioni addizionali per ottenere una maggiore efficienza delle operazioni