

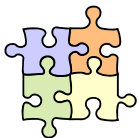
# Architetture Software

## Broker

**Dispensa ASW 420**  
ottobre 2014

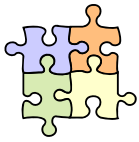
*Tutti sanno che una certa cosa  
è impossibile da realizzare,  
finché arriva uno sprovveduto che non lo sa  
e la inventa.*

*Albert Einstein*



## - Fonti

- [POSA1] Pattern-Oriented Software Architecture, 1996
- [POSA4] Pattern-Oriented Software Architecture – A Pattern Language for Distributed Computing, 2007



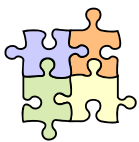
## - Obiettivi e argomenti

### □ Obiettivi

- introdurre i pattern POSA per gli stili di comunicazione distribuita
- presentare il pattern architetturale Broker

### □ Argomenti

- infrastruttura per la distribuzione
- Broker [POSA]
- discussione



## - Wordle





## \* Infrastruttura per la comunicazione

- Lo sviluppo di sistemi software distribuiti è sostenuto da opportuni strumenti di **middleware**
  - i componenti di un sistema software distribuito possono essere focalizzati sulla logica di business dell'applicazione
  - il collegamento tra questi componenti avviene mediante connettori – che comunicano sulla base di stili di comunicazione distribuita implementata dal middleware
  - il middleware è un'infrastruttura di comunicazione che protegge i componenti delle applicazioni dalle numerose complessità tipiche dei sistemi distribuiti



## Middleware e stili di comunicazione

- Sviluppare middleware per la comunicazione è un'attività estremamente complessa
  - fortunatamente, solo raramente c'è la necessità di progettare e implementare nuovi stili di comunicazione
  - infatti, sono disponibili un'ampia varietà di piattaforme di middleware di comunicazione, standard e commerciali, già usate con successo in moltissimi sistemi distribuiti
- Malgrado le differenze nei dettagli, le tecnologie di middleware sostengono di solito uno o più di tre stili di comunicazione distribuita differenti – rappresentati da tre pattern architetturali fondamentali
  - invocazione di metodi remoti – **Broker** [POSA]
  - messaging – **Messaging** [POSA]
  - comunicazione publish-subscribe – **Publisher-Subscriber** [POSA]



# Stili di comunicazione e pattern [POSA]

## □ *Broker* [POSA]

- organizza il sistema distribuito in un insieme di componenti che interagiscono sulla base di invocazioni di metodi remoti
- il broker (o una federazione di broker) gestisce gli aspetti della comunicazione tra questi oggetti distribuiti

## □ *Messaging* [POSA]

- organizza il sistema in componenti che interagiscono sulla base dello scambio di messaggi
- i messaggi (che possono codificare dati, metadati, documenti, richieste, risposte, notifica di eventi) vengono scambiati tramite un insieme di canali per messaggi interconnessi

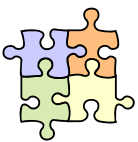
## □ *Publisher-Subscriber* [POSA]

- organizza il sistema in componenti che interagiscono scambiandosi la notifica di eventi in modo asincrono

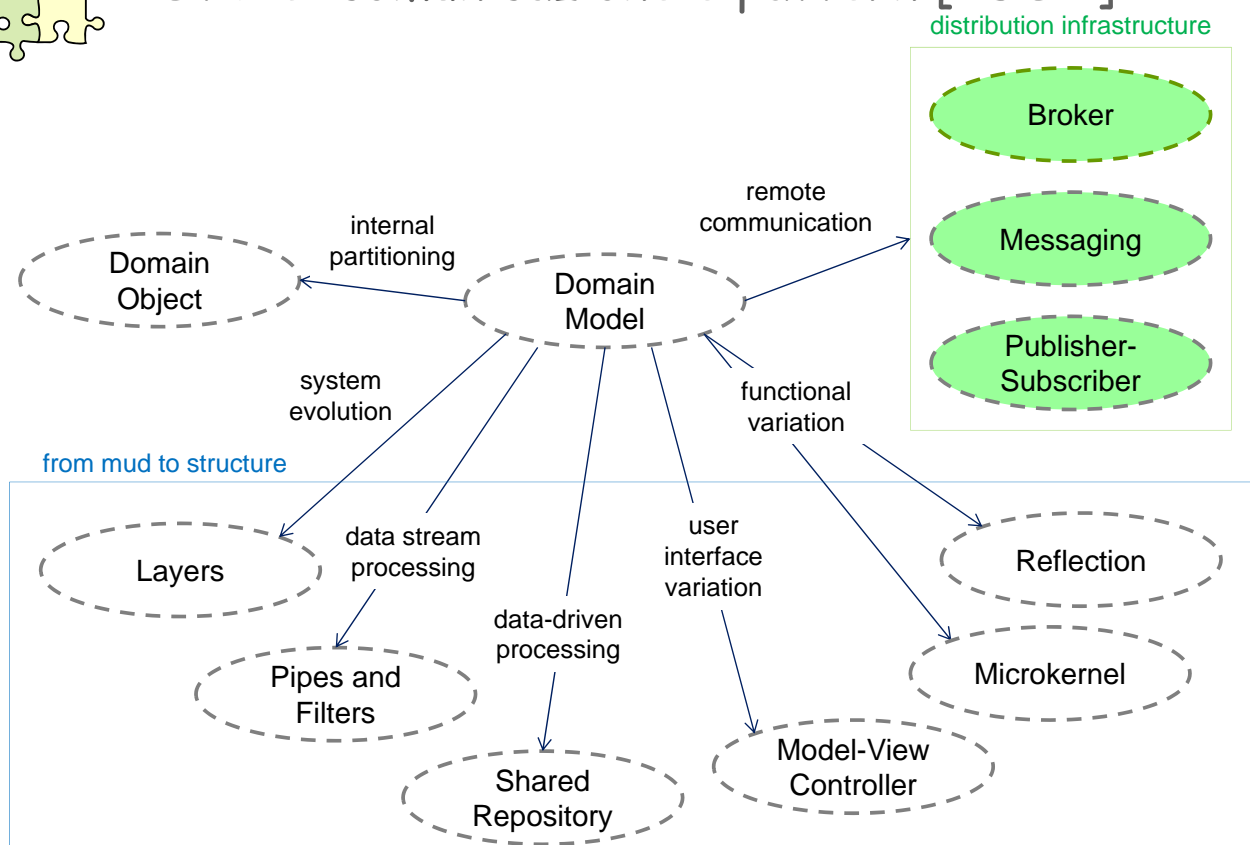
7

Broker

Luca Cabibbo - ASw



# Stili di comunicazione e pattern [POSA]



8

Broker

Luca Cabibbo - ASw



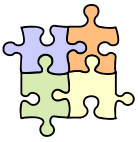
## Stili di comunicazione e middleware

- Nel seguito questi stili di comunicazione verranno discussi separatamente
  - Broker in questa dispensa, Messaging e Publisher-Subscriber in una dispensa successiva
  - in pratica, ciascuno strumento di middleware implementa spesso uno o più di questi stili di comunicazione – ad esempio, nelle tecnologie a componenti e nelle architetture orientate ai servizi

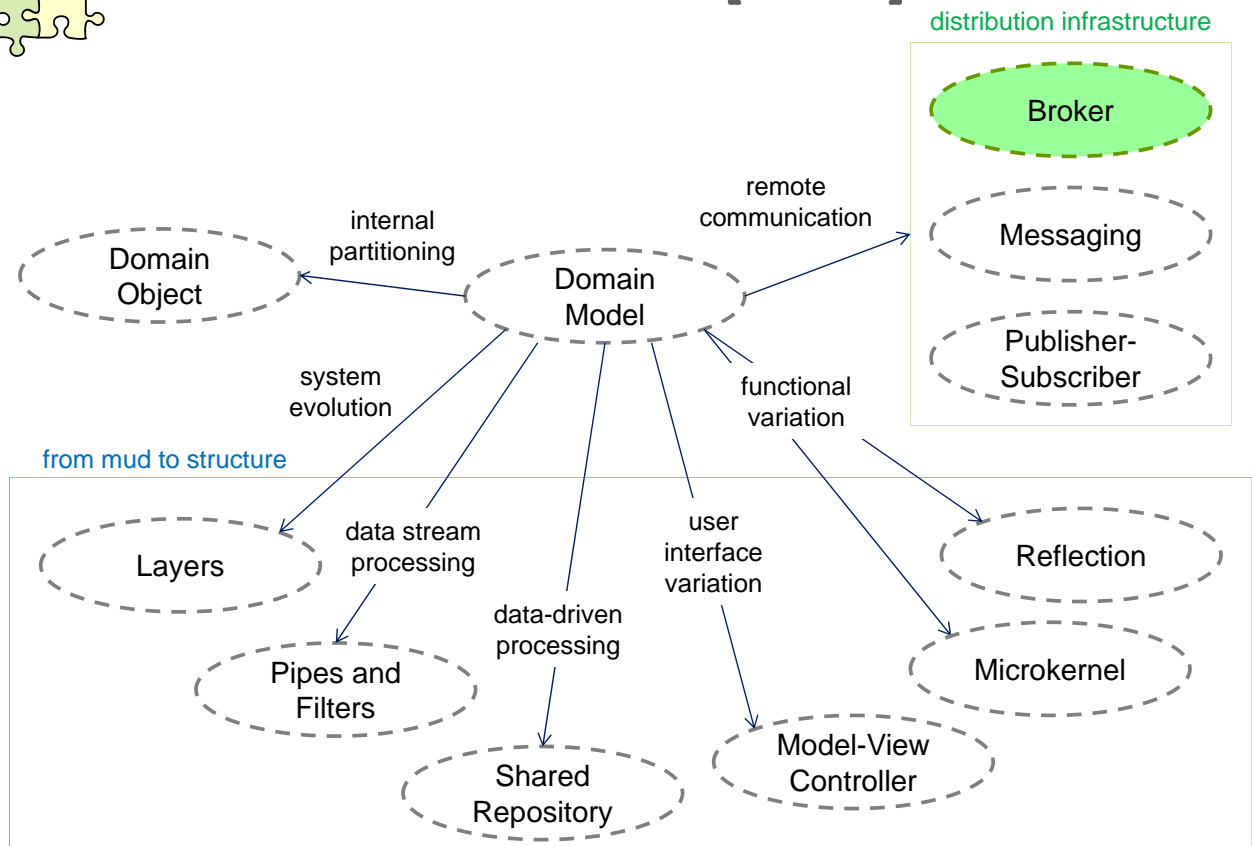


## \* Broker [POSA]

- Il pattern architetturale **Broker**
  - può essere usato per strutturare sistemi distribuiti con componenti disaccoppiati che interagiscono mediante l'invocazione di servizi remoti
  - un elemento broker è responsabile del coordinamento della comunicazione tra i diversi componenti remoti
    - ad es., per inoltrare richieste e trasmettere risposte o eccezioni
- In generale, il termine *broker* indica un *intermediario*
  - ad esempio, una parte che media tra un acquirente e un venditore
  - “un professionista che ricerca e acquista, per conto del cliente, nel mercato di riferimento, il prodotto che offre il miglior rapporto qualità-prezzo” [Wikipedia]



## Relazione con altri stili [POSA]



11

Broker

Luca Cabibbo - ASw



## Broker

- [POSA4] colloca Broker nella categoria "infrastrutture per la distribuzione"
  - Broker è un pattern alla base di molte tecnologie a oggetti/servizi distribuiti
- costituisce un'infrastruttura di comunicazione per rendere trasparenti all'applicazione (e al programmatore) alcune complessità della distribuzione

12

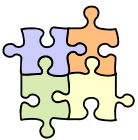
Broker

Luca Cabibbo - ASw



## Esempio

- City Information System – CIS
  - sistema di informazioni turistiche
  - portale verso altri sistemi (esterni) che effettivamente offrono servizi per turisti
    - informazioni su alberghi e ristoranti, trasporti pubblici, musei, visite guidate, ...
    - in alcuni casi con la possibilità di fare prenotazioni/acquisti
  - ci possono essere più sistemi esterni che possono soddisfare uno stesso tipo di richieste
  - è possibile la registrazione dinamica di nuovi sistemi esterni
  - il CIS si propone come un punto di contatto singolo per il turista nei confronti dei sistemi esterni
    - dunque, CIS è un “broker” tra turista e sistemi esterni



## Broker

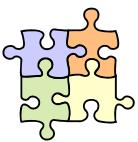
- Contesto
  - ci sono più componenti in grado di erogare dei servizi – in un ambiente distribuito di erogatori di servizi
  - questi componenti potrebbero essere eterogenei



# Broker

## □ Problema

- si vuole organizzare il sistema in modo che sia possibile utilizzare un insieme di componenti distribuiti, in modo flessibile – in modo che gli utenti dei servizi non debbano conoscere la natura e la posizione dei fornitori dei servizi – e in modo che sia possibile cambiare dinamicamente i collegamenti tra utenti e fornitori di servizi
  - componenti interoperabili ma disaccoppiati
  - trasparenza nell'accesso ai componenti – ad es. di locazione
  - possibilità di aggiungere/rimuovere/sostituire componenti a runtime
- in generale, le applicazioni distribuite dovrebbero gestire le problematiche connesse alla distribuzione usando un modello di programmazione che protegga le applicazioni stesse dai dettagli della rete e della posizione dei componenti in rete



# Broker

## □ Soluzione

- introduci un componente intermediario *broker* – per avere un disaccoppiamento tra utenti (client) e fornitori (server) dei servizi
  - in un sistema distribuito, il broker incapsula l'infrastruttura di comunicazione
  - inoltre, il broker definisce un modello di programmazione distribuita in cui i client possono richiedere servizi remoti come se fossero servizi locali
  - in questo modo, i dettagli della comunicazione sono separati dalle funzionalità applicative
- utilizza degli ulteriori intermediari *proxy* – per aiutare i componenti client e server nella gestione dei dettagli dell'interazione con il broker

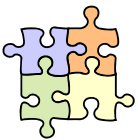




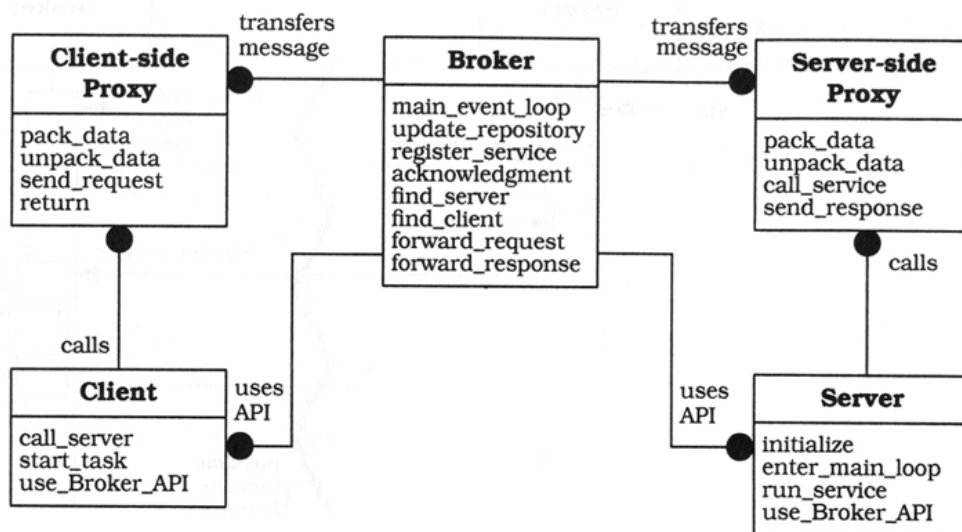
# Broker

## □ Soluzione

- dinamica dell'interazione
  - i server registrano i propri servizi presso il broker
  - un client accede ai servizi indirettamente, tramite il broker – il broker inoltra ciascuna richiesta di un client ad un server opportuno, e poi trasmette al client la risposta ricevuta – in questo modo i client possono ignorare l'identità, la locazione e le caratteristiche del server che li stanno servendo
- se un server diviene indisponibile, il broker può scegliere dinamicamente di sostituirlo con un altro server (che offre un servizio compatibile)
  - il broker è l'unico componente che deve sapere di questo cambiamento – senza ripercussioni sui client



# Struttura (parziale)





## Partecipanti (1)

### □ *Server*

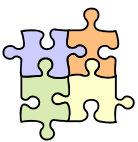
- ciascun server è un oggetto o componente che offre servizi
- i servizi sono esposti tramite una interfaccia – ad es., IDL
- di solito ci sono molti server, che offrono gli stessi servizi o anche servizi diversi

### □ *Client*

- ciascun client è un oggetto (o componente o applicazione) che vuole fruire di servizi
- di solito ci sono molti client concorrenti
- ciascun client inoltra le proprie richieste al Broker

### □ Client e server non nell'accezione (stretta) dello stile client/server

- ma nell'accezione DOA – in una certa interazione, un oggetto/componente client vuole fruire di un servizio fornito da un oggetto/componente server



## Partecipanti (2)

### □ *Broker*

- come abbiamo detto, il termine “broker” indica un intermediario
- il broker è un bus – un “messaggero” – responsabile della trasmissione di richieste e risposte tra client e server
- offre ai server (mediante delle API) la funzionalità per registrare i loro servizi
- offre ai client (mediante delle API) la funzionalità per richiedere l'esecuzione di servizi
- può offrire altri servizi aggiuntivi – ad es., naming (directory, context)



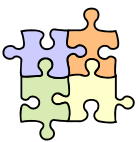
## Partecipanti (3)

### □ *Proxy lato client*

- intermediario tra client e broker/server
  - è il rappresentante lato client del servizio richiesto – con la stessa interfaccia del servizio
- vive nel processo del client
- un remote proxy – fornisce trasparenza rispetto alla distribuzione, perché sia il broker che l'oggetto server remoto appaiono locali al client

### □ *Proxy lato server*

- intermediario tra broker e server
- vive nel processo del server
- responsabile di ricevere richieste dal client (tramite il broker), di invocare il servizio effettivo e di trasmettere le risposte al client (tramite il broker)



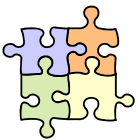
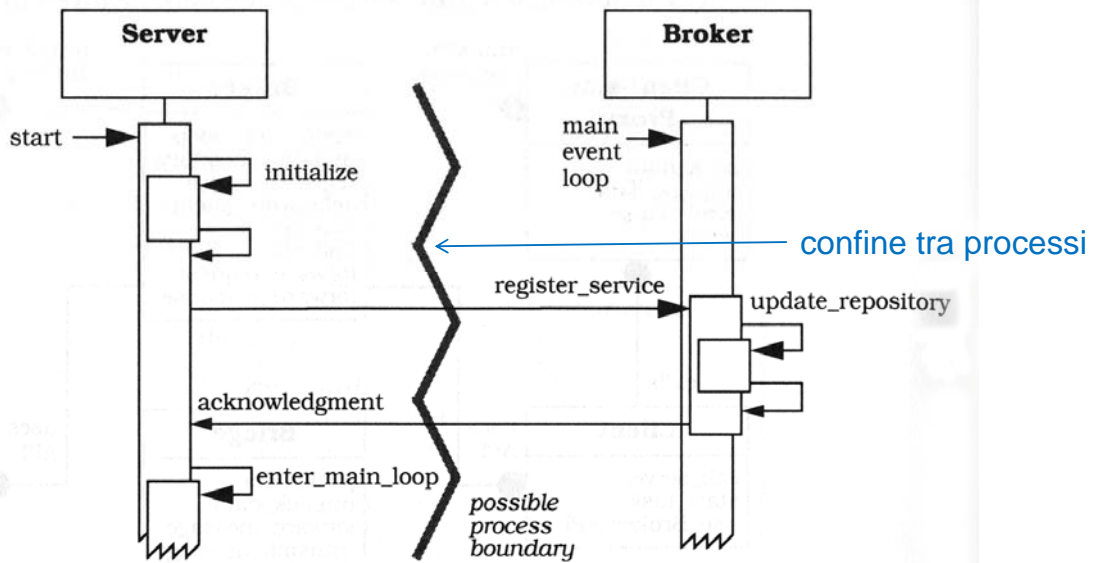
## Parentesi: Proxy [POSA, GoF]

- Il design pattern **Proxy** fa comunicare i client di un componente che offre servizi con un rappresentante di quel componente – e non direttamente con il componente stesso
  - l'introduzione di un tale “segnaposto” può sostenere diversi scopi – ad es., aumentare l'efficienza, semplificare l'accesso, consentire la protezione da accessi non autorizzati
- Proxy [GoF]
  - fornisce un surrogato o un segnaposto per un altro oggetto – per controllarne l'accesso
- Possibili diverse applicazioni di proxy
  - remote proxy, protection proxy, cache proxy, synchronization proxy, virtual proxy, ...
  - in Broker, si tratta di un remote proxy



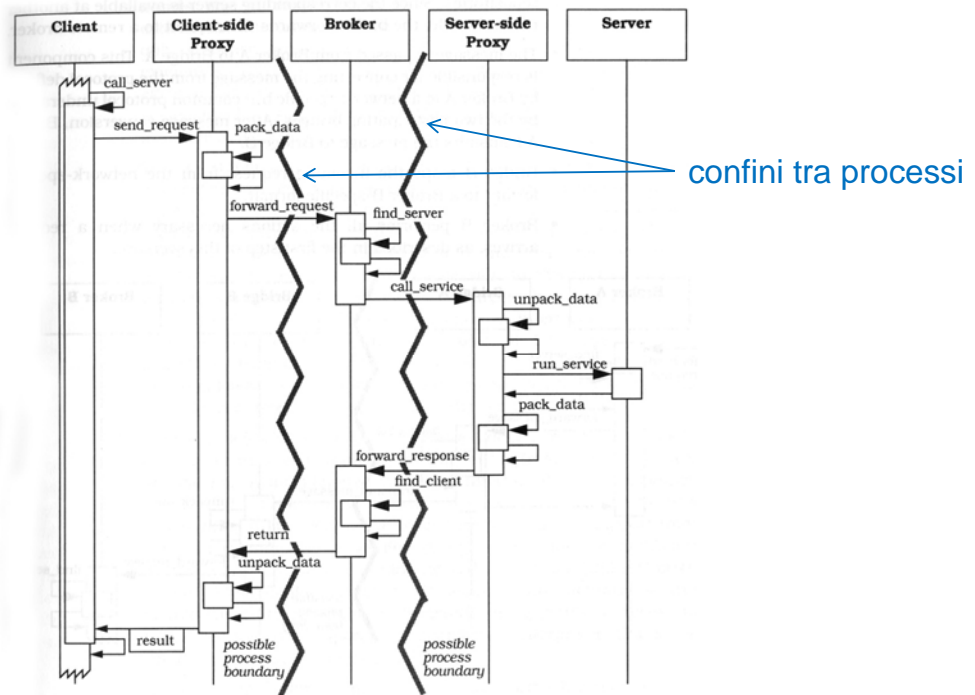
## Scenario 1

- Registrazione di un server presso un broker



## Scenario 2

- Gestione di una richiesta da parte di un client





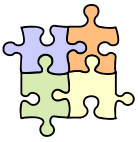
## Scenario 2

- Ruolo del broker in questo scenario 2 – trascurando il ruolo dei proxy (che già conosciamo)
  - il client chiede l'erogazione di un servizio – ma non la chiede direttamente a un server, ma piuttosto la chiede al broker
  - il broker (sulla base delle informazioni sui servizi registrati, che gestisce direttamente) seleziona un server in grado di erogare il servizio richiesto
    - potrebbero esserci un solo server in grado di erogare quel servizio, ma potrebbero essercene anche più di uno
  - inoltre, il broker inoltra la richiesta al server selezionato, per conto del client
  - poi, il broker ottiene la risposta alla richiesta dal server, sempre per conto del client
  - infine, il broker restituisce la risposta al client



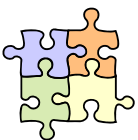
## Scenario 2 - varianti

- Lo scenario per la gestione delle richieste da parte di un client ha due varianti principali – nelle due varianti, il broker ha ruoli diversi – ma anche i proxy hanno ruoli un po' diversi
  - **comunicazione indiretta**
    - come mostrato dallo scenario 2 di base, *tutte* le richieste (e le risposte) transitano attraverso il broker
    - il client e il server *non* comunicano *mai* in modo diretto
  - **comunicazione diretta**
    - il broker è responsabile solo di mettere in comunicazione client e server
    - dopo di che, client e server comunicano in modo diretto – richiede che client e server comprendano e utilizzino lo stesso protocollo

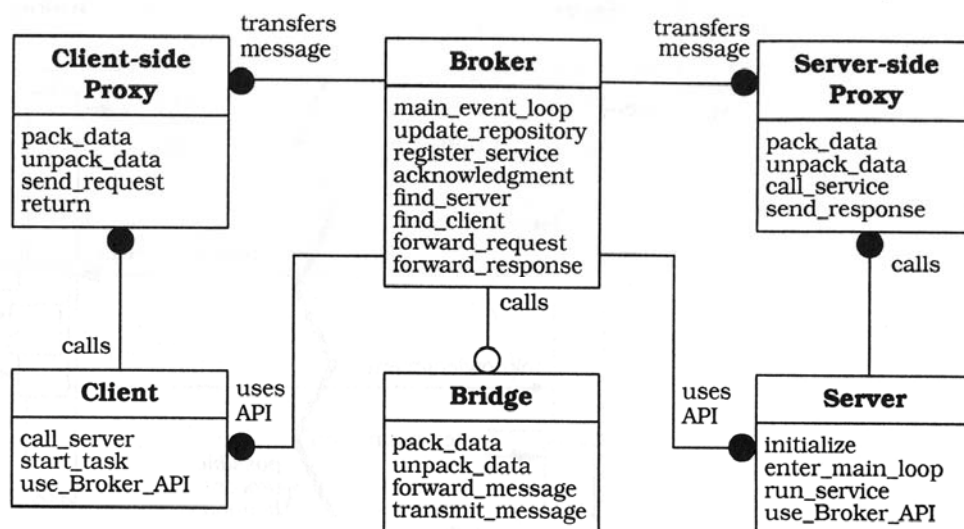


## Scenario 2 - varianti

- Confronto tra le due varianti
  - nella comunicazione diretta, l'overhead di comunicazione è minore
  - tuttavia, nella comunicazione indiretta, il client è protetto in modo continuo da eventuali indisponibilità e variazioni di locazione dei servizi
  - inoltre la comunicazione indiretta abilita lo scenario 3 – uno scenario di interoperabilità, relativo alla soluzione completa proposta da Broker



## Struttura (completa)

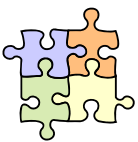




## Partecipanti (4)

### □ *Bridge*

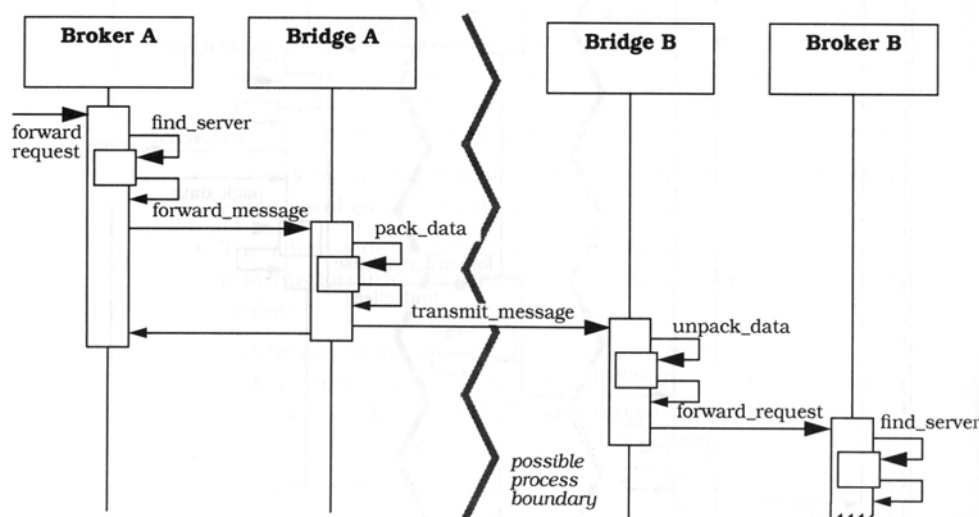
- si tratta di un componente opzionale – per collegare/far interoperare più broker
- in particolare, ciascun broker è di solito specializzato per una particolare “tecnologia” – ma può essere dotato di un proprio bridge per interoperare con altri broker – con riferimento a un qualche protocollo per l’interoperabilità
- di solito, in un sistema basato su broker
  - si utilizza un broker per ciascuna tecnologia che deve interoperare
  - ciascun broker è associato a un proprio bridge
  - i diversi bridge interagiscono tra di loro sulla base di un protocollo per l’interoperabilità che è indipendente dalle specifiche tecnologie utilizzate dai singoli broker



## Scenario 3

### □ Interazione tra broker multipli

- ciascun broker gestisce componenti “omogenei”
- i diversi broker comunicano mediante bridge

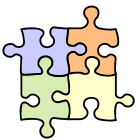




## Conseguenze

### □ Benefici

- 😊 trasparenza dalla posizione – il client non ha bisogno di sapere dove si trova il server – i server possono ignorare la posizione dei loro client
- 😊 modificabilità ed estendibilità dei componenti
- 😊 interoperabilità tra tipi di client, server e broker diversi
- 😊 riusabilità di servizi esistenti



## Conseguenze

### □ Inconvenienti

- 😞 riduzione delle prestazioni
- 😞 minor tolleranza ai guasti rispetto a una soluzione non distribuita – il broker potrebbe essere un punto di fallimento singolo (ma se è opportunamente replicato/ridondato, allora potrebbe non esserlo)
- 😞 testing difficile

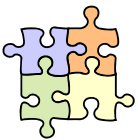




## Esempio: Java RMI

### □ Java RMI

- l'architettura Java RMI è sostanzialmente basata su Broker
  - il ruolo del broker è svolto dalla JVM remota che contiene l'oggetto servente
  - il ruolo del proxy lato client è svolto dalla JVM lato client – e da uno stub generato implicitamente
  - il ruolo del proxy lato server (lo skeleton) è svolto dalla JVM lato server – anche in questo caso, lo skeleton è generato implicitamente
- la comunicazione Java RMI è normalmente basata sul protocollo JRMP
  - tuttavia, la tecnologia Java RMI-IIOP (Java RMI over Internet Inter-ORB Protocol) consente di interoperare con dei broker CORBA – si tratta di una “tecnologia bridge”



## - Broker e tattiche per la modificabilità

### □ *Increase semantic coherence – increase cohesion*

- ciascun server raggruppa un insieme di funzionalità semanticamente correlate

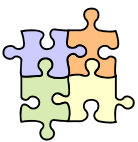
### □ *Encapsulate – reduce coupling*

- ciascun server espone le proprie funzionalità mediante un'interfaccia remota pubblicata – l'interazione con i client avviene solo sulla base di tale interfaccia
- anche il broker espone le proprie responsabilità pubbliche (ad es., relativamente al registry dei servizi) mediante delle API pubbliche – in modo indipendente dall'implementazione di tali responsabilità



## Broker e tattiche per la modificabilità

- *Use an intermediary – reduce coupling*
  - il broker è un intermediario tra client e server – fornisce la possibilità di invocare servizi in modo indipendente dalla locazione
  - anche il proxy lato client è un intermediario – gli oggetti remoti appaiono locali al client
  - anche il proxy lato server è un intermediario – il server riceve delle chiamate locali
  - la coppia di proxy nasconde a client e server dettagli implementativi della comunicazione remota



## Broker e tattiche per la modificabilità

- *Restrict dependencies – reduce coupling*
  - il client comunica con il server indirettamente, tramite i proxy ed il broker – questo consente cambiamenti nei server motivati, ad esempio, da obiettivi di affidabilità o bilanciamento del carico
- *Use runtime binding – defer binding*
  - i server si registrano presso il broker a runtime – la registrazione può cambiare dinamicamente



## \* Discussione

- Le tecnologie di middleware utilizzate nella realizzazione dei sistemi distribuiti sostengono di solito tre stili principali di comunicazione distribuita – rappresentati da tre pattern architetturali fondamentali
  - in questa dispensa abbiamo studiato Broker
  - in una successiva dispensa studieremo Messaging e Publisher-Subscriber
  - in pratica, ciascuno strumento di middleware implementa spesso uno o più di questi stili di comunicazione



## Discussione

- Middleware per l'invocazione di metodi remoti
  - consente la realizzazione di protocolli sincroni di tipo richiesta-risposta
- Middleware per il messaging
  - consente lo scambio asincrono di messaggi
  - sostiene accoppiamento debole, flessibilità ed affidabilità



## Discussione

- Middleware per componenti
  - i componenti vivono in contenitori (application server) in grado di gestire la configurazione e la distribuzione dei componenti, e fornire ad essi funzionalità di supporto
  - possibile sia la comunicazione sincrona che asincrona
  
- Middleware orientato ai servizi
  - enfasi sull'interoperabilità tra componenti eterogenei, sulla base di protocolli standard aperti ed universalmente accettati
  - possibile sia la comunicazione sincrona che asincrona
  - flessibilità nell'organizzazione dei suoi elementi (servizi)
  
- In continua evoluzione