OO e Distribuzione

- Sistemi Distribuiti Basati sugli Oggetti
- Basati sugli Oggetti e non Orientati agli Oggetti
- Modello degli Attori

Nozioni sui Sistemi ad Oggetti

Oggetto:

Entità che incapsula delle informazioni e le operazioni ammesse su tali informazioni. Interagisce con altri oggetti tramite la propria interfaccia.

Classe:

È il tipo di dato astratto che permette di descrivere le proprietà degli oggetti, le loro operazioni e la loro interfaccia.

Ereditarietà:

È il meccanismo che permette di definire nuove classi a partire da quelle esistenti specificando solo come differiscono da esse.

Nozioni sui
Sistemi ad Oggetti (Segue)

<u>Delegazione</u>:

È il meccanismo che permette ad un oggetto di delegare l'esecuzione di un servizio ad un altro oggetto.

Peter Wegner ha classificato i linguaggi che dispongono del concetto di classe e oggetto come <u>object−based</u> e se dispongono anche dell'ereditarietà come <u>object−oriented</u>.

Il meccanismo della delegazione può essere utilizzato come un sostituto per l'ereditarietà in linguaggi object−based.

Sistemi Distribuiti
Basati sugli Oggetti

Un sistema distribuito basato sugli oggetti fornisce le caratteristiche di un sistema basato sugli oggetti operante in ambiente distribuito.

- **Distribuzione**
- **Trasparenza (es. accesso uniforme, localizzazione)**
- **Tolleranza ai guasti**
- **Disponibilità**

Struttura degli Oggetti

La struttura degli oggetti influenza la progettazione dell'intero sistema.

- **Granularità**
- **Composizione**

Struttura degli Oggetti (Granularitá)

La dimensione, lo spreco e l'ammontare delle elaborazioni eseguite da un oggetto caratterizzano la sua granularità.

- **Oggetti a grana grossa**

  Oggetto di grandi dimensioni (codice) che tende a fare tutto. Ed ha pochissime interazioni con gli altri oggetti.

  Sono semplici ed hanno uno spazio di indirizzamento proprio, ma poco flessibili e pesanti per lo scheduling del sistema.

- **Oggetti a grana media**

  Codice di medie dimensioni. Molti oggetti a grana media occupano lo spazio di indirizzamento di un oggetto a grana grossa. Permettono di aumentare la concorrenza.

  Svantaggio dovuto all'aumento dell'overhead.

Struttura degli Oggetti (Granularitá Segue)

- **Oggetti a grana fine**

  Oggetto di piccole dimensioni (codice) con molte e frequenti interazioni con gli altri oggetti.

  Sintetici ma richiedono una invocazione di metodo da o verso altri oggetti per ogni azione che intraprendono.

Gli oggetti sono incapsulati in base alla grana:

grossa $\subset$ media $\subset$ fine

Struttura degli Oggetti (Composizione)

La relazione tra processi e oggetti caratterizza la composizione degli oggetti. Due possibili modelli:

- **Ad oggetti passivi;**
- **Ad oggetti attivi.**

Nota: Gli oggetti non sono supportati a livello di SO come entità autonome.

Composizione
(Modello Passivo)

I processi e gli oggetti (passivi) sono entità completamente separate.

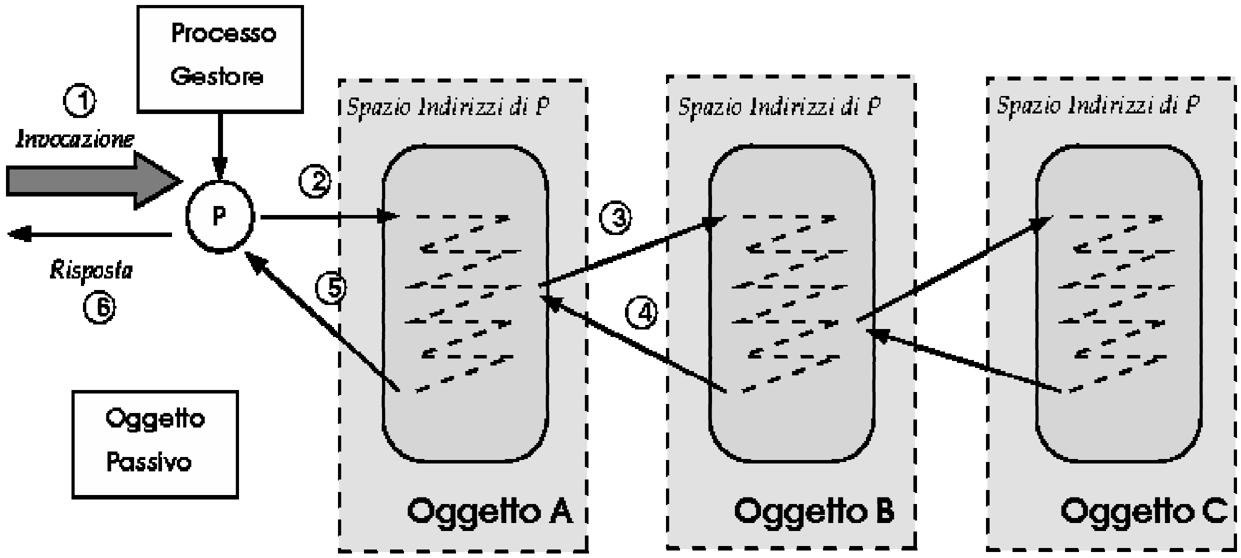
Ogni processo non è legato ad un solo oggetto, bensì è legato al compimento di una singola azione.

Il processo gestore si "muove" da un oggetto all'altro.

Vantaggio: illimitato numero di processi legati ad un singolo oggetto.

Svantaggio: elevato overhead dovuto al "muoversi" dell'oggetto .

Composizione
(Modello Passivo)



Composizione
(Modello Attivo)

Uno o più processi (*worker*) sono associati ad ogni oggetto e gestiscono le richieste che gli vengono inoltrate.

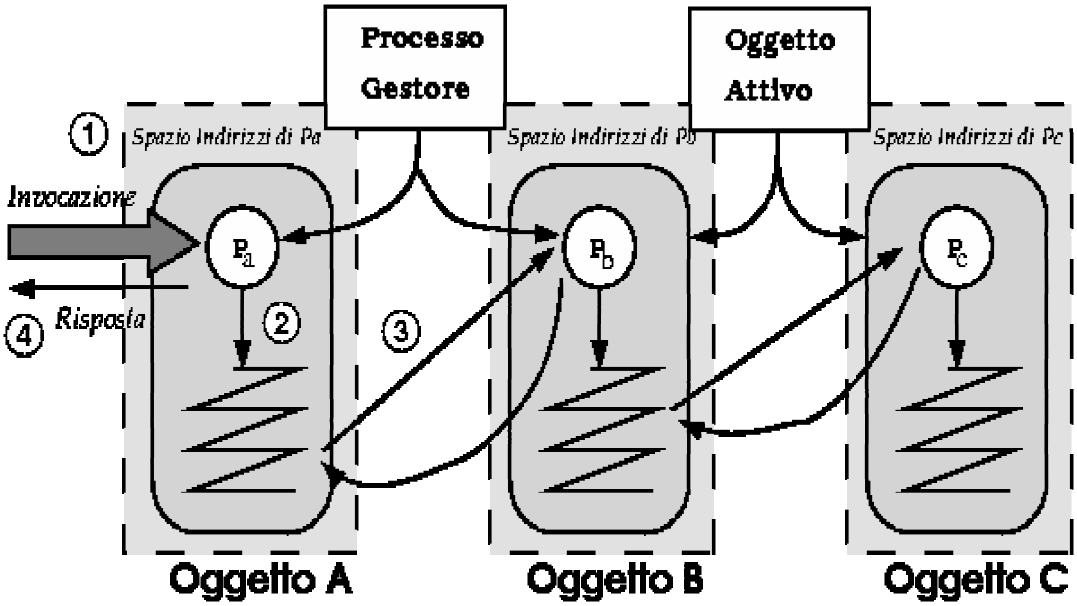
Il lifecycle di un processo è legato al lifecycle dell'oggetto a cui è associato.

Più processi vengono usati per portare a termine una computazione.

Se le richieste pendenti superano i worker di un certo oggetto o sono inserite in una coda in attesa di essere processate o si crea un nuovo processo per gestirle.

<u>Nota</u>: è possibile incorrere in deadlock.

Composizione
(Modello Attivo)



Gestione degli Oggetti

In un sistema distribuito basato sugli oggetti, gli oggetti sono le entità portanti della computazione. Pertanto il sistema deve:

- Sincronizzare le chiamate ad uno specifico oggetto
- Garantire la consistenza dello stato di un oggetto
- Proteggere gli oggetti da accessi indesiderati
- Recuperare da eventuali fallimenti

La maggioranza delle tecniche sono recuperate dalle tecniche usate nei sistemi distribuiti classici.

Gestione delle Azioni

Le azioni devono godere delle seguenti proprietà:

- Serializzabilità

  Azioni concorrenti devono essere schedulate in modo che l'effetto complessivo sia come se fossere eseguite sequenzialmente in un qualche ordine.

- Atomicità

  Un'azione o termina con successo o non ha nessun effetto.

- Permanenza

  L'effetto di un'azione che termina con successo non va perso.

Un'azione può coinvolgere più oggetti e richiedere l'invocazione di molti metodi.

Gestione delle Azioni

Un’azione termina con successo (*commit*) solo se tutte le invocazioni di metodi che coinvolge terminano con successo.

Solo se un’azione commit, tutte le modifiche fatte agli oggetti diventano effettive. Se un’azione fallisce allora tutti i cambiamenti relativi vengono disfatti.

<u>Nota</u>: La procedura di commitment garantisce che tutte le modifiche fatte ad un oggetto siano rese permanenti o che nessuna lo sia.

Gestione delle Azioni
(Two–phase Commit Protocol)

- **Una richiesta di precommit è inoltrata a tutti gli oggetti coinvolti.**
- **Ricevuta la richiesta, un oggetto salva le proprie modifiche e ritorna un acknowledge.**
- **Se tutti gli oggetti mandano il loro acknowledge in un tempo prefissato:**
    - **Si inoltra una richiesta di commit , altrimenti**
    - **Si inoltra una richiesta di abort.**
- **Sul commit gli oggetti rendono permanenti le modifiche e confermano il commit. Sull’abort scartano le modifiche.**
- **La richiesta di commit viene iterata fintanto che tutti gli oggetti confermano l’avvenuto commit.**

Nota c’è rischio di deadlock.

Gestione delle Azioni
(Segue)

Quando iniziare una fase di commit?

- **Su richiesta.**
  È l'utente che si occupa di iniziare una fase di commit.
  Maggiore efficienza.
- **Ad ogni azione (transition scheme).**
  È il sistema ad effettuare una fase di commit.
  Garantita l'atomicità e la persistenza delle azioni.

Gestione delle Azioni
(Sincronizzazione)

Si deve assicurare che le chiamate ad uno stesso oggetto non interferiscano l'un l'altra.

Un meccanismo di sincronizzazione serve per garantire la proprietà di serializzazione e l'integrità di un oggetto.

Due schemi principali:

- **Schema di sincronizzazione pessimistico**
- **Schema di sincronizzazione ottimistico**

Sincronizzazione

(schema di sincronizzazione pessimistico)

- **Il sistema previene i conflitti.**

  Un'azione che invoca un oggetto è temporaneamente sospesa nel caso possa interferire con un'altra azione è che correntemente servita dall'oggetto.

  Quando l'azione che genera conflitto termina con successo, l'azione sospesa viene ripresa.

  Per la sincronizzazione si usano lock di lettura/scrittura o monitor e semafori.

Sincronizzazione

(schema di sincronizzazione ottimistico)

- **Il sistema <u>non</u> previene i conflitti, tenta di risolverli se accadono.**

  Prima che un'azione possa dare il commit viene eseguito un test per verificare che le azioni possano essere serializzate.

  Solo se verifica che le modifiche non sono in conflitto con quelle fatte da un altro oggetto che ha già dato il commit provvederà a dare il commit.

  Gli oggetti sono replicati permettendo un più alto grado di concorrenza.

  Azioni che terminano con successo possono essere abortite perchè in conflitto con altre. A scapito delle performance.

Gestione degli Oggetti
(Sicurezza)

Previene il fatto che clienti non autorizzati accedano a servizi offerti da un oggetto.

Diversi meccanismi, i principali:

- **Capabilities**

  Una capability è una chiave composta da un nome e da un insieme di diritti.

  Il nome identifica un oggetto, mentre l'insieme dei diritti specifica quali servizi di quell'oggetto si ha diritto di invocare.

  Le capability sono passate ai clienti e saranno un parametro delle richieste.

- **Procedura di controllo**

  L'oggetto che riceve una richiesta, controlla che tale richiesta possa essere soddisfatta. Più flessibile.

Gestione degli Oggetti
(Affidabilitá)

Un sistema distribuito basato sugli oggetti deve essere in grado di rilevare e risolvere il fallimento di uno o più degli oggetti che gestisce.

- **Object recovery**
  - **Roll−back recovery scheme**

    Gli oggetti sono fatti ripartire dall'ultimo stato stabile.

  - **Roll−forward recovery scheme**

    Si ripetono le azioni interrotte.

- **Replicazione degli oggetti**
  - **Oggetti immutabili**
  - **Replicazione a copia primaria (*statica* e *dinamica*)**

    Modifiche fatte solo alla copia primaria

  - **Peer objects (tutte copie primarie)**

Gestione delle Interazioni tra Oggetti

Compito fondamentale di un middleware object−based consiste nel gestire le interazioni tra oggetti appartenenti ad un sistema distribuito.

- **Localizzare gli oggetti remoti**
- **Gestire la comunicazione**
- **Rilevare il fallimento degli oggetti**

Interazioni tra Oggetti
(Localizzazione)

Gli oggetti non sanno dove sono fisicamente gli altri oggetti con cui vogliono interagire. Sarà il middleware sottostante a fornire un meccanismo di *<u>localizzazione trasparente</u>*.

Gli oggetti sono localizzati tramite degli <u>identificatori unici</u> ed <u>immutabili</u> che vengono assegnati loro dal middleware stesso.

- **Oid**
- **DNS (completo e parziale)**
- **Cache−broadcast**
- **Forward location pointer**

Interazioni tra Oggetti
(gestione delle invocazioni)

Il middleware responsabile di effettuare i passi necessari per consegnare ogni richiesta al giusto destinatario e ritornare i risultati al richiedente.

La gestione delle invocazioni dipende dal modello supportato, due schemi:

- **Scambio di messaggi**
- **Invocazione diretta**

Gestione delle Invocazioni
(scambio di messaggi)

Un middleware basato sul modello ad oggetti attivi supporta lo scambio di messaggi.

La richiesta viene impacchettata in un messaggio e spedita ad un server process dell'oggetto destinatario.

Il server process valuta se accettare la richiesta e nell'ipotesi spacchetta il messaggio. Il risultato viene impacchettato in un altro messaggio e rispedito al mittente.

Vantaggio: supporta facilmente la mobilità degli oggetti

Svantaggio: overhead inutile per comunicazioni sulla stessa macchina.

Gestione delle Invocazioni
(Invocazione Diretta)

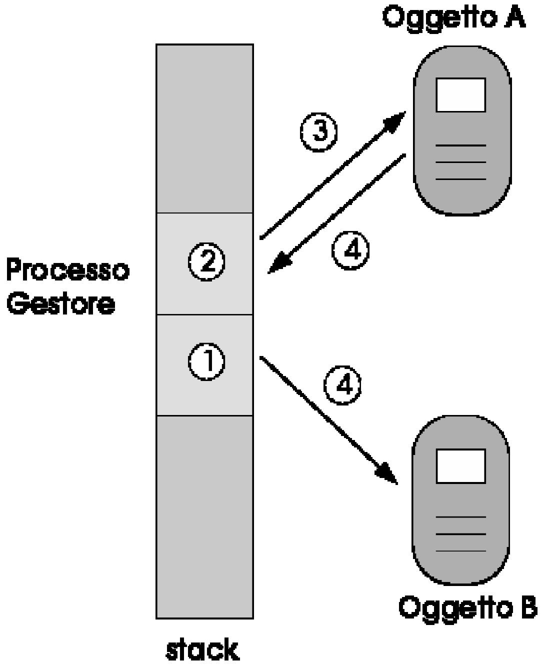
Un middleware basato sul modello ad oggetti passivi supporta lo schema ad invocazione diretta.

Nel modello ad oggetti passivi un solo processo si occupa di gestire tutta un'azione, migrando di oggetto in oggetto quando una richiesta viene effettuata.

Due casi:

- **Richiesta per un oggetto sullo stesso computer**
- **Richiesta per un oggetto su un computer remoto**

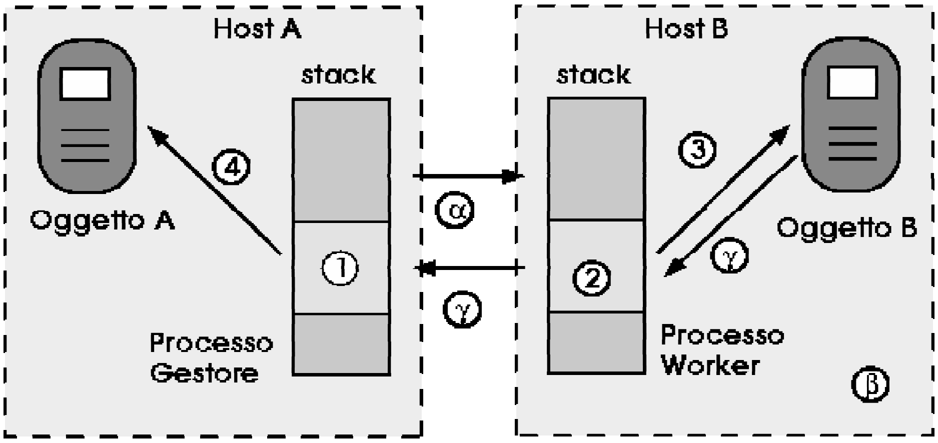
Gestione delle Invocazioni
(Invocazione Diretta – Chiamata Locale)



1. **Si salvano sullo stack le informazioni relative all'oggetto che si sta lasciando.**
2. **I parametri della richiesta vengono posti sullo stack.**
3. **L'oggetto invocato viene caricato in memoria e la chiamata è fatta partire.**
4. **Quando l'operazione termina i risultati sono ritornati al chiamante e lo stato viene ripristinato a prima della chiamata.**

Gestione delle Invocazioni

(Invocazione Diretta – Chiamata Remota)



1. Si salvano sullo stack le informazioni relative all'oggetto che si sta lasciando.

   α. Un messaggio con i dati relativi alla richiesta viene creato ed inviato al computer remoto.

   β. Il computer ricevente crea un processo worker per gestire la richiesta.

2. I parametri della richiesta vengono posti sullo stack.
3. L'oggetto invocato viene caricato in memoria e la chiamata è fatta partire.

   γ. Si crea un messaggio contente i risultati, lo si invia al computer richiedente, il processo worker viene distrutto.

4. Quando l'operazione termina i risultati sono ritornati al chiamante e lo stato viene ripristinato a prima della chiamata.

Riferimenti Bibliografici

- **Distributed Object–Based Programming Systems. Roger Chin and Samuel Chanson. In ACM Computer Surveys, Vol. 23(1), March 1991.**
- **Analysis of Inheritance Anomaly in Object–Oriented Concurrent Programming Language. Satoshi Matsuoka, and Akinori Yonezawa. In G. Agha, P. Wegner, and A. Yonezawa Editors Research Directions in OO Concurrent Programming. MIT Press. 1993.**