



Middleware a oggetti distribuiti

Corso di Laurea in Informatica, Programmazione Distribuita
Delfina Malandrino, dmalandrino@unisa.it
<http://www.unisa.it/docenti/delfinamalandrino>



Organizzazione della lezione

2

- Cosa è il Middleware a oggetti distribuiti
 - Motivazioni
 - Tipi di middleware a oggetti distribuiti
- L'evoluzione del Middleware
 - Il progenitore: Remote Procedure Call
 - Da RPC al Middleware a oggetti Distribuiti
- Alcuni esempi di Middleware
 - Corba
 - Microsoft .NET (con Remoting)
 - Enterprise Java (con Java RMI)
- Middleware implicito ed esplicito
- Il ruolo degli oggetti distribuiti nel Middleware
- Conclusioni

Organizzazione della lezione

3

- Cosa è il Middleware a oggetti distribuiti
 - Motivazioni
 - Tipi di middleware a oggetti distribuiti
- L'evoluzione del Middleware
 - Il progenitore: Remote Procedure Call
 - Da RPC al Middleware a oggetti Distribuiti
- Alcuni esempi di Middleware
 - Corba
 - Microsoft .NET (con Remoting)
 - Enterprise Java (con Java RMI)
- Middleware implicito ed esplicito
- Il ruolo degli oggetti distribuiti nel Middleware
- Conclusioni

Gli oggetti distribuiti

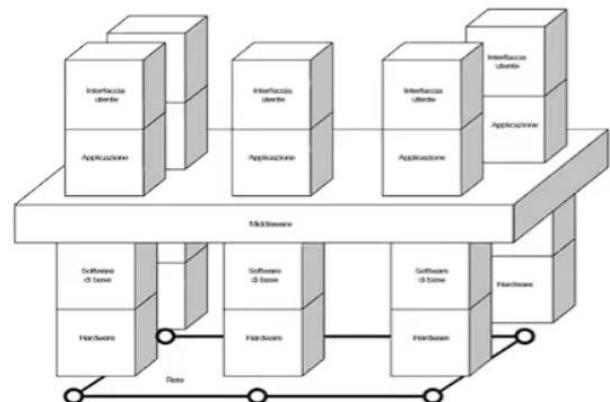
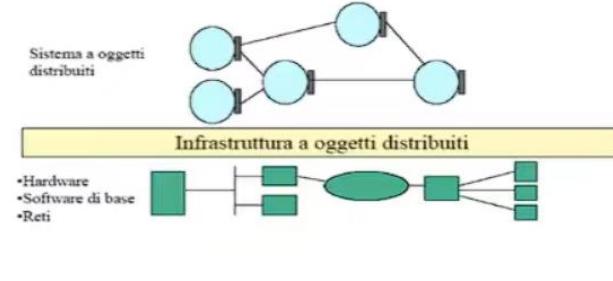
4

- Gli oggetti distribuiti si trovano alla confluenza di due aree della tecnologia software:
 - i sistemi distribuiti
 - che puntano a realizzare un unico sistema integrato basato sulle risorse offerte da diversi calcolatori messi in rete
 - lo sviluppo e la programmazione orientata agli oggetti
 - che si focalizzano sulle modalità per ridurre la complessità dei sistemi software, creando artefatti software riutilizzabili in diversi contesti
- I sistemi distribuiti basati su Oggetti Distribuiti sono uno degli strumenti utilizzati dai sistemi distribuiti per assicurare
 - estendibilità
 - affidabilità
 - scalabilità
 - rendendo minimo lo sforzo per progettare, sviluppare e manutenere sistemi complessi

Come si realizza questa integrazione?

5

- La risposta è: tramite il Middleware!
- Uno strato software che si trova tra:
 - Applicazioni
 - Sistema operativo, protocolli di rete e Hardware



A chi serve il middleware?

6

- È chiaro che possiamo programmare un sistema distribuito utilizzando le primitive di comunicazione a disposizione di ogni singolo nodo
- Questo risulta essere:
 - complesso: necessario risolvere i dettagli di interoperabilità a basso livello per ogni coppia di macchine nel sistema distribuito
 - costoso: tempo e risorse necessarie per lo sviluppo (per ogni piattaforma SW/HW serve un esperto nel team)
 - dettagli di basso livello come trasformare strutture dati del livello applicazione in stream di byte oppure datagram che possono essere trasmessi su rete
- Scopo del middleware: rendere semplici tutti questi compiti, e fornire le astrazioni appropriate per i programmati
 - che ben si integrano con gli strumenti tradizionali che usano per sviluppare l'applicazione



Organizzazione della lezione

7

- Cosa è il Middleware a oggetti distribuiti
 - Motivazioni
 - Tipi di middleware a oggetti distribuiti
- L'evoluzione del Middleware
 - Il progenitore: Remote Procedure Call
 - Da RPC al Middleware a oggetti Distribuiti
- Alcuni esempi di Middleware
 - Corba
 - Microsoft .NET (con Remoting)
 - Enterprise Java (con Java RMI)
- Conclusioni

Tipi di Middleware

8

- Middleware di **infrastruttura**
 - Si occupa della comunicazione tra S.O. diversi e della gestione della concorrenza in maniera da essere portabile (un esempio è la Java Virtual Machine)
- Middleware di **distribuzione**: automatizza compiti comuni per la comunicazione come
 - marshalling: invio di parametri per le invocazioni remote (complesso per la eterogeneità)
 - multiplexing dello stesso canale di comunicazione per più invocazioni
 - gestione della semantica delle invocazioni (unicast, multicast, attivazione on-demand)
 - riconoscimento e gestione malfunzionamenti
- Middleware **per servizi comuni**
 - persistenza, transazioni, sicurezza, etc

L'astrazione fornita dal Middleware

9

- L'accesso alle risorse, mediate dai tre livelli di middleware permette di focalizzarsi sullo sviluppo dell'applicazione
- Favorisce il riuso delle soluzioni adottate (che non dipendono dall'hardware/software sottostante ma dal middleware utilizzato)
- Rendono lo sviluppo efficace ed efficiente

Organizzazione della lezione

10

- Cosa è il Middleware a oggetti distribuiti
 - Motivazioni
 - Tipi di middleware a oggetti distribuiti
- L'evoluzione del Middleware
 - Il progenitore: Remote Procedure Call
 - Da RPC al Middleware a oggetti Distribuiti
- Alcuni esempi di Middleware
 - Corba
 - Microsoft .NET (con Remoting)
 - Enterprise Java (con Java RMI)
- Conclusioni

Remote Procedure Call

11

- Meccanismo di base per il dialogo di applicazioni distribuite (1984)
- Basato su C, l'idea era di permettere l'invocazione di procedure remote come se fossero locali
- L'obiettivo:
 - facilitare il compito del progettista/programmatore
 - che può concentrarsi sulla suddivisione delle funzionalità in procedure
 - senza curarsi del fatto che siano remote o locali
- Torneremo su questo approccio quando parleremo di Remote Method Invocation

Le innovazioni di RPC

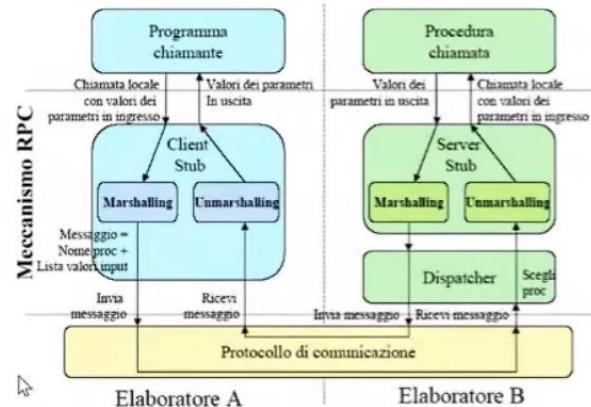
12

- Prima tecnologia a fornire la traduzione dei tipi di dato a livello applicazione
 - trasmissione dei dati attraverso stream di byte su socket, in modo che fossero automaticamente codificati/decodificati (procedura di marshalling)
 - superamento delle differenze nella rappresentazione di interi (con le differenze hardware tra architetture big-endian e little-endian), e stringhe (ASCII vs. EBCDIC)
- Permettere al programmatore di usare un paradigma noto: sincronia della invocazione forzata
 - client bloccato finché il server non produce la risposta richiesta

Le innovazioni di RPC

13

- Utilizzo di client stub e server stub per forzare marshalling, rappresentazione dei dati e sincronia... .
- ... creati automaticamente attraverso un linguaggio specifico, chiamato *Interface Definition Language (IDL)*



Invocazioni sincrone

14

- Classico paradigma dell'invocazione di funzioni (metodi)
- La funzione chiamante (processo, thread, ...) viene bloccata fino a quando la esecuzione della funzione (metodo) remota non ha terminato
- ... ed ha restituito il risultato
- Classica semantica di programmazione non-concorrente (familiare e di semplice utilizzo)
- Invocazioni asincrone: quando la funzione (processo, thread, ...) chiamante continua la computazione, concorrentemente alla computazione della funzione chiamata
 - possibile l'uso di code di smistamento di messaggi, possibile la non contemporanea presenza di client e server

Alcuni problemi e difficoltà di RPC

15

- Innanzitutto, paradigma procedurale
 - ▣ e non con il paradigma a oggetti!
- I tipi di dato sono solamente elementari
 - ▣ limitazioni per tipi di dato composti (struct) e/o puntatori
- Mancanza della gestione delle eccezioni malfunzionamento sul canale che non blocca la computazione
- Invocazione concorrente di più programmi

Organizzazione della lezione

16

- Cosa è il Middleware a oggetti distribuiti
 - ▣ Motivazioni
 - ▣ Tipi di middleware a oggetti distribuiti
- L'evoluzione del Middleware
 - ▣ Il progenitore: Remote Procedure Call
 - ▣ Da RPC al Middleware a oggetti Distribuiti
- Alcuni esempi di Middleware
 - ▣ Corba
 - ▣ Microsoft .NET (con Remoting)
 - ▣ Enterprise Java (con Java RMI)
- Conclusioni

Il passaggio da RPC a Oggetti Distribuiti

17

- Gli oggetti distribuiti sono un modello presentato negli anni '90 che unisce la tecnologia software della programmazione ad oggetti con quella dei sistemi distribuiti:
 - il modello RPC viene esteso in maniera da permettere l'invocazione di metodi di oggetti remoti
- Ma tale estensione...

Il passaggio da RPC a Oggetti Distribuiti

18

- ... non è banale come sembra
- Ai meccanismi già realizzati, si deve aggiungere (almeno!)
 - polimorfismo
 - ereditarietà
 - gestione delle eccezioni
- In un certo senso, il passaggio che ha portato dal modello RPC al modello ad oggetti distribuiti può essere visto come un ulteriore passo della tecnologia dei linguaggi di programmazione nel cammino verso l'incapsulamento, la modularità e l'astrazione
 - corrispondente distribuito del passo in avanti nei linguaggi di programmazione passando dal paradigma procedurale (C, Pascal, ...) a quello orientato a oggetti (Java, C++, C#, ...)

Le motivazioni al passaggio

19

- Evoluzione di sistemi distribuiti complessi, difficili da:
 - progettare (complessità ed ampiezza)
 - manutenere
 - fare evolvere
- Chiave di volta per realizzare sistemi
 - eterogenei (diverse rappresentazione dei dati)
 - scalabili
 - tolleranti ai malfunzionamenti estendibili
 - di facile gestione
- In effetti, la tecnologia degli oggetti distribuiti ha rappresentato, negli anni '90, la chiave di volta per realizzare sistemi distribuiti eterogenei che fossero scalabili, tolleranti ai malfunzionamenti, estendibili e di agevole gestione

Organizzazione della lezione

20

- Cosa è il Middleware a oggetti distribuiti
 - Motivazioni
 - Tipi di middleware a oggetti distribuiti
- L'evoluzione del Middleware
 - Il progenitore: Remote Procedure Call
 - Da RPC al Middleware a oggetti Distribuiti
- Alcuni esempi di Middleware
 - Corba
 - Microsoft .NET (con Remoting)
 - Enterprise Java (con Java RMI)
- Conclusioni

Middleware CORBA

21

- Common Object Request Broker Architecture (CORBA) è uno standard, proposto nel 1991, che permette ad oggetti distribuiti, scritti in diversi linguaggi e quindi eterogenei, di comunicare e collaborare per realizzare una applicazione distribuita
 - metodo remoto invocato su un oggetto distribuito
- Multilinguaggio
 - C, C++, Java, ma anche Cobol, Ada, ...
- Interoperabilità garantita dal binding con CORBA e dalla specifica dei servizi indipendenti dal linguaggio (IDL)
- Object Request Broker
 - si occupa di fornire diversi tipi di trasparenza a invocazione
 - ma anche di fornire accesso a servizi di supporto (anche evoluti)
- Ambiente complesso (anche per il contesto in cui è stato progettato) e con qualche problema di interoperabilità tra fornitori diversi

Organizzazione della lezione

22

- Cosa è il Middleware a oggetti distribuiti
 - Motivazioni
 - Tipi di middleware a oggetti distribuiti
- L'evoluzione del Middleware
 - Il progenitore: Remote Procedure Call
 - Da RPC al Middleware a oggetti Distribuiti
- Alcuni esempi di Middleware
 - Corba
 - Microsoft .NET (con Remoting)
 - Enterprise Java (con Java RMI)
- Conclusioni

.NET Framework

23

- .NET Framework è la soluzione Microsoft per la realizzazione di applicazioni
 - basato su Common Language Runtime (CLR)
 - macchina virtuale che esegue applicazioni scritte in uno dei linguaggi Microsoft (C#, Visual Basic, F# etc.)
- .NET Remoting eredita il ruolo di meccanismo di comunicazione remota tra oggetti avuto da DCOM e COM+ in passato
- Suddivisione dei compiti (sui tre middleware):
 - Remoting si occupa della comunicazione
 - CLR si occupa della infrastruttura
 - servizi assicurati da altre librerie Microsoft

Organizzazione della lezione

24

- Cosa è il Middleware a oggetti distribuiti
 - Motivazioni
 - Tipi di middleware a oggetti distribuiti
- L'evoluzione del Middleware
 - Il progenitore: Remote Procedure Call
 - Da RPC al Middleware a oggetti Distribuiti
- Alcuni esempi di Middleware
 - Corba
 - Microsoft .NET (con Remoting)
 - Enterprise Java (con Java RMI)
- Conclusioni

Enterprise Java

25

- Obiettivo: limitare la complessità per realizzare nuovi servizi basati su oggetti
facilitandone il riuso
- Netta separazione del layer di presentazione da quello di business
 - in modo da permettere politiche di bilanciamento di carico attraverso le traparenze di migrazione, locazione, accesso, etc.
- Modello a componenti

Enterprise Java: componenti

26

- Una Componente Distribuita è un blocco riutilizzabile di software che può essere combinato in un sistema distribuito per realizzare funzionalità
- All'interno di una componente risiedono servizi e applicazioni che espongono tramite una interfaccia le proprie funzionalità
- Quello che caratterizza e differenzia le componenti da altri moduli software riutilizzabili (come gli oggetti, ad esempio)
 - è che essi possono essere combinati sotto forma di eseguibili binari, piuttosto che sotto forma di azioni da compiere sul codice sorgente
 - il modello a componenti si basa sul cosiddetto middleware implicito che viene contrapposto alle tecnologie di middleware esplicito (come sono tutte quelle descritte finora)

Organizzazione della lezione

27

- Cosa è il Middleware a oggetti distribuiti
 - Motivazioni
 - Tipi di middleware a oggetti distribuiti
- L'evoluzione del Middleware
 - Il progenitore: Remote Procedure Call
 - Da RPC al Middleware a oggetti Distribuiti
- Alcuni esempi di Middleware
 - Corba
 - Microsoft .NET (con Remoting)
 - Enterprise Java (con Java RMI)
- **Middleware implicito ed esplicito**
- Il ruolo degli oggetti distribuiti nel Middleware
- Conclusioni

Enterprise Java: middleware implicito ed esplicito

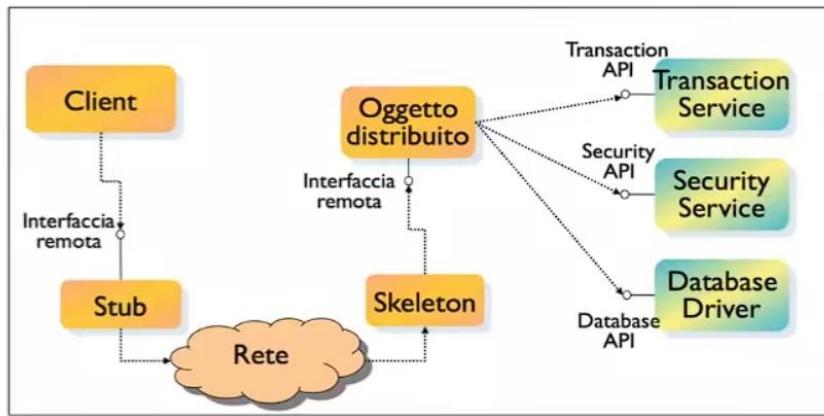
28

- Il middleware implicito, attraverso meccanismi di intercettazione delle richieste e delle interazioni tra gli oggetti, è in grado di fornire servizi comuni e trasversali ad ogni componente
 - senza che essa debba esplicitamente richiederli all'interno del codice
- Il server che gestisce la componente (detta *application server* o *container*) fornisce questi servizi sulla base delle richieste (codificate non nel codice ma in un file di metadati di descrizione) specificate quando la componente viene messa a disposizione sul server (*fase di deployment*)
- In questa maniera i servizi vengono messi a disposizione in maniera completamente trasparente allo sviluppatore di software della componente, realizzando una maggiore interoperabilità tra produttori di software diversi

Middleware esplicito

30

- Sun Java RMI, CORBA, DCOM
- uso dei servizi in maniera esplicita



Un esempio di Middleware esplicito

31

```
...
public void transfer(Account acct1, Account acct2,
    long amount) {
    //1.call Middleware API per security check
    //2.call Middleware API per iniziare una
        transazione
    //3.call Middleware API per caricare i dati
    //4.sottrai amount da account1 e somma
        amount in account2
    //5.call Middleware API per memorizzare i
        dati
    //6.call Middleware API per finire la
        transazione
}
```

Logica di business

Un esempio di Middleware esplicito

32

```
//...
public void transfer(Account acct1, Account acct2,
    long amount) {
    //1.call Middleware API per security check ←
    //2.call Middleware API per iniziare una
        transazione

    //3.call Middleware API per caricare i dati

    //4.sottrai amount da account1 e somma
        amount in account2

    //5.call Middleware API per memorizzare i
        dati

    //6.call Middleware API per finire la
        transazione
}
```

Logica di business

Middleware...

Un esempio di Middleware esplicito

33

```
//...
public void transfer(Account acct1, Account acct2,
    long amount) {
    //1.call Middleware API per security check
    //2.call Middleware API per iniziare una
        transazione←

    //3.call Middleware API per caricare i dati

    //4.sottrai amount da account1 e somma
        amount in account2

    //5.call Middleware API per memorizzare i
        dati

    //6.call Middleware API per finire la
        transazione
}
```

Logica di business

Middleware...

prima ...

Un esempio di Middleware esplicito

34

```
//...
public void transfer(Account acct1, Account acct2,
    long amount) {
    //1.call Middleware API per security check
    //2.call Middleware API per iniziare una
        transazione
    //3.call Middleware API per caricare i dati ← ...
    //4.sottrai amount da account1 e somma
        amount in account2
    //5.call Middleware API per memorizzare i
        dati
    //6.call Middleware API per finire la
        transazione
}
```

Logica di business

Middleware...

prima ...

...

Un esempio di Middleware esplicito

35

```
//...
public void transfer(Account acct1, Account acct2,
    long amount) {
    //1.call Middleware API per security check
    //2.call Middleware API per iniziare una
        transazione
    //3.call Middleware API per caricare i dati
    //4.sottrai amount da account1 e somma
        amount in account2
    //5.call Middleware API per memorizzare i
        dati ← ...
    //6.call Middleware API per finire la
        transazione
}
```

Logica di business

Middleware...

prima ...

...

Middleware...

Un esempio di Middleware esplicito

36

```
//...
public void transfer(Account acct1, Account acct2,
    long amount) {
    //1.call Middleware API per security check
    //2.call Middleware API per iniziare una
        transazione
    //3.call Middleware API per caricare i dati
    //4.sottrai amount da account1 e somma
        amount in account2
    //5.call Middleware API per memorizzare i
        dati
    //6.call Middleware API per finire la
        transazione
}
```

Logica di business

Middleware...

prima...

...

Middleware...

...dopo

Un esempio di Middleware esplicito

37

```
//...
public void transfer(Account acct1, Account acct2,
    long amount) {
    //1.call Middleware API per security check
    //2.call Middleware API per iniziare una
        transazione
    //3.call Middleware API per caricare i dati
    //4.sottrai amount da account1 e somma
        amount in account2
    //5.call Middleware API per memorizzare i
        dati
    //6.call Middleware API per finire la
        transazione
}
```

Logica di business

Middleware...

prima...

...

Middleware...

...dopo

Difficile da
scrivere e da
manutenere
(cambia MW ⇒
cambio API)

Come vorremmo che fosse

38

```
//...
public void transfer(Account acct1, Account acct2,
    long amount) {
    //sottrai amount da account1 e somma amount
    //in account2
}
```

Solo logica di business

Come vorremmo che fosse

39

```
//...
public void transfer(Account acct1, Account acct2,
    long amount) {
    //sottrai amount da account1 e somma amount
    //in account2
}
```

Solo logica di business

Obiettivo: usare il
middleware senza
conoscerne le API

Come vorremmo che fosse

40

```
//...
public void transfer(Account acct1, Account
    acct2, long amount) {
    //sottrai amount da account1 e somma amount
    //in account2
}
```

Solo logica di business

Obiettivo: usare il middleware senza conoscerne le API

Middleware implicito o dichiarativo

- › codice dell'oggetto distribuito (senza MW)

Come vorremmo che fosse

41

```
//...
public void transfer(Account acct1, Account
    acct2, long amount) {
    //sottrai amount da account1 e somma amount
    //in account2
}
```

Solo logica di business

Obiettivo: usare il middleware senza conoscerne le API

Middleware implicito o dichiarativo

- › codice dell'oggetto distribuito (senza MW)
- › dichiarazione dei servizi di middleware da usare (deployment descriptor)

Come vorremmo che fosse

42

```
//...
public void transfer(Account acct1, Account
    acct2, long amount) {
    //sottrai amount da account1 e somma amount
    //in account2
}
```

Solo logica di business

Obiettivo: usare il middleware senza conoscerne le API

Middleware implicito o dichiarativo

- › codice dell'oggetto distribuito (senza MW)
- › dichiarazione dei servizi di middleware da usare (deployment descriptor)

in un file di testo XML: *"Per favore, vorrei sicurezza, persistenza e transazioni!"*

Come vorremmo che fosse

43

```
//...
public void transfer(Account acct1, Account
    acct2, long amount) {
    //sottrai amount da account1 e somma amount
    //in account2
}
```

Solo logica di business

Obiettivo: usare il middleware senza conoscerne le API

Middleware implicito o dichiarativo

- › codice dell'oggetto distribuito (senza MW)
- › dichiarazione dei servizi di middleware da usare (deployment descriptor)
 - in un file di testo XML: *"Per favore, vorrei sicurezza, persistenza e transazioni!"*
- › compilazione del deployment descriptor

Come vorremmo che fosse

44

```
//...
public void transfer(Account acct1, Account
    acct2, long amount) {
    //sottrai amount da account1 e somma amount
    //in account2
}
```

Solo logica di business

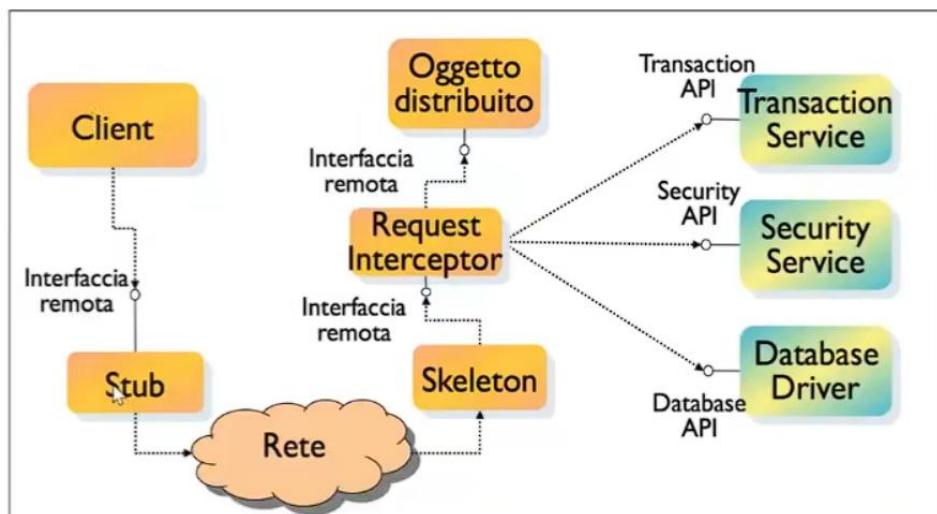
Obiettivo: usare il middleware senza conoscerne le API

Middleware implicito o dichiarativo

- › codice dell'oggetto distribuito (senza MW)
- › dichiarazione dei servizi di middleware da usare (deployment descriptor)
 - in un file di testo XML: *"Per favore, vorrei sicurezza, persistenza e transazioni!"*
- › compilazione del deployment descriptor
 - con un tool del MW vendor: genera un request interceptor

Middleware implicito

45



Organizzazione della lezione

46

- Cosa è il Middleware a oggetti distribuiti
 - Motivazioni
 - Tipi di middleware a oggetti distribuiti
- L'evoluzione del Middleware
 - Il progenitore: Remote Procedure Call
 - Da RPC al Middleware a oggetti Distribuiti
- Alcuni esempi di Middleware
 - Corba
 - Microsoft .NET (con Remoting)
 - Enterprise Java (con Java RMI)
- Middleware implicito ed esplicito
- **Il ruolo degli oggetti distribuiti nel Middleware**
- Conclusioni

Il ruolo degli oggetti distribuiti... concludendo

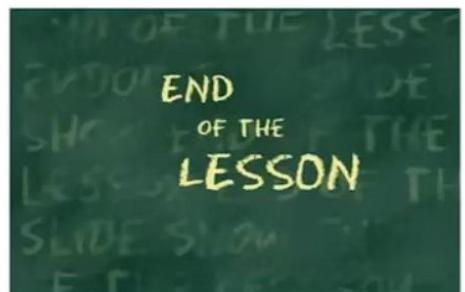
47

- Modello a componenti (Enterprise Computing): architetture basate su blocco riutilizzabile di software, ricombinabili semplicemente per ottenere funzionalità più complesse
- Gli oggetti distribuiti assicurano l'interoperabilità tra componenti in un ambiente eterogeneo (sempre più, pensare al mobile!)
- Collegamento ideale tra la programmazione anni 80 (socket) e quella anni 2000 (servizi) punto cruciale per assicurare la scalabilità, la tolleranza ai malfunzionamenti, la estendibilità e la facilità di gestione
- “Protocollo di comunicazione” assicurato dalla interoperabilità tra oggetti distribuiti (Java RMI, .Net Remoting, etc.)
 - permettendo la remotizzazione di layer della architettura

Conclusioni

48

- Cosa è il Middleware a oggetti distribuiti
 - Motivazioni
 - Tipi di middleware a oggetti distribuiti
- L'evoluzione del Middleware
 - Il progenitore: Remote Procedure Call
 - Da RPC al Middleware a oggetti Distribuiti
- Alcuni esempi di Middleware
 - Corba
 - Microsoft .NET (con Remoting)
 - Enterprise Java (con Java RMI)
- Middleware implicito ed esplicito
- Il ruolo degli oggetti distribuiti nel Middleware
- Conclusioni



Nelle prossime lezioni:

Programmazione concorrente e thread