Corso di Laurea in Informatica
III Anno Triennale
Programmazione Distribuita – classe 1

Open Distributed Processing Reference Model e Trasparenza

Delfina Malandrino
dmalandrino@unisa.it
http://www.unisa.it/docenti/delfinamalandrino

1

Organizzazione della lezione

- Un modello di riferimento per i Sistemi Distribuiti
  - Cosa è un modello di riferimento
  - Open Distributed Processing
- Le caratteristiche di un sistema distribuito
- I requisiti non funzionali di un sistema distribuito
- La trasparenza di un sistema distribuito
- Conclusioni

2

Organizzazione della lezione

- Un modello di riferimento per i Sistemi Distribuiti
  - Cosa è un modello di riferimento
  - Open Distributed Processing
- Le caratteristiche di un sistema distribuito
- I requisiti non funzionali di un sistema distribuito
- La trasparenza di un sistema distribuito
- Conclusioni

3

Organizzazione della lezione

- Allo scopo di facilitare lo sviluppo dei sistemi distribuiti, è importante la condivisione di un modello comune, che serva come astrazione comune per:
  - produttori (hw/sw)
  - progettisti
  - sviluppatori
- Indipendente dalla specifica implementazione, tecnologia
  - ma sufficientemente dettagliato ed informativo sui meccanismi ed i metodi/funzionalità implementati
- Fornisca anche un terreno comune per la comunicazione durante le fasi iniziali della progettazione:
  - identificando termini e linguaggio per la definizione del sistema (comune a comunità diverse)
  - permettendo confronti tra diverse implementazioni di sistemi reali

4

Definizione

- The Reference Model of Open Distributed Processing (RM-ODP)
  - Modello dell'ISO/IEC per la standardizzazione dei sistemi distribuiti (4 documenti di specifica)
- Obiettivo:
  - *"Favorire la diffusione dei benefici della distribuzione di servizi di elaborazione di informazione in un ambiente eterogeneo (nodi e risorse) in multipli domini amministrativi di gestione"*
  - Basato sul modello ISO/OSI a 7 livelli
    - si innesta sul settimo livello (application)
- Espande il modello ISO/OSI
  - concentrandosi sulla **comunicazione** piuttosto che sulla semplice **connessione** inglobando concetti più ad alto livello (portabilità, trasparenza, etc.)

5

Organizzazione della lezione

- Un modello di riferimento per i Sistemi Distribuiti
  - Cosa è un modello di riferimento
  - Open Distributed Processing
- Le caratteristiche di un sistema distribuito
- I requisiti non funzionali di un sistema distribuito
- La trasparenza di un sistema distribuito
- Conclusioni

6

Organizzazione della lezione

- Le caratteristiche (viste nella precedente lezione)
  - Remoto
  - Concorrenza
  - Assenza di uno stato globale
  - Malfunzionamenti parziali
  - Eterogeneità
  - Autonomia
  - Evoluzione
  - Mobilità

7

Organizzazione della lezione

- Un modello di riferimento per i Sistemi Distribuiti
  - Cosa è un modello di riferimento
  - Open Distributed Processing
- Le caratteristiche di un sistema distribuito
- I requisiti non funzionali di un sistema distribuito
- La trasparenza di un sistema distribuito
- Conclusioni

8

Requisiti non funzionali - 1

- Definizione:
  - Non direttamente collegati alle funzionalità che deve realizzare il sistema distribuito
    - Non parte dei requisiti funzionali
    - Specificano la qualità del sistema, da un punto di vista globale

9

Requisiti non funzionali - 2

- Sistemi distribuiti *aperti*
  - uso di interfacce e standard noti e riconosciuti
    - per facilitare l'interoperabilità e l'evoluzione
    - per evitare di rimanere legati ad un singolo fornitore:
      - se si usano standard aperti, si può cambiare fornitore senza particolari rischi per l'intera architettura (che può essere riutilizzata ed integrata)
- Sistemi distribuiti *integrati*
  - per incorporare al proprio interno sistemi e risorse differenti senza dover utilizzare strumenti ad-hoc
    - si assicura eterogeneità hardware, software e delle applicazioni

10

Requisiti non funzionali - 3

- Sistemi distribuiti *flessibili*
  - per far evolvere i sistemi distribuiti in maniera da integrare sistemi *legacy* al proprio interno
  - per gestire modifiche durante l'esecuzione
    - accomodare cambi a run-time, riconfigurandosi dinamicamente
- Sistemi distribuiti *modulari*
  - ogni componente autonoma ma interdipendente verso il resto del sistema

11

Requisiti non funzionali - 4

- Sistemi distribuiti che *supportino la federazione di sistemi*
  - unione di diversi sistemi (amministrativamente e architetturalmente)
  - per fornire servizi in maniera congiunta
- Sistemi distribuiti *facilmente gestibili*
  - in modo da permettere controllo, gestione e manutenzione per configurarne
    - i servizi
    - la loro quality of service
    - le politiche di accesso

12

12

Requisiti non funzionali - 4

- Garantire *Qualità dei servizi (QoS)* allo scopo di fornire servizi con vincoli di tempo, disponibilità e affidabilità anche in situazioni di malfuzionameni parziali
  - La tolleranza ai malfunzionamenti è una delle principali richieste di qualità del servizio di un sistema distribuito
    - I sistemi centralizzati sono particolarmente poco tolleranti ai malfunzionamenti, che possono rendere l'intero sistema inutilizzabile
    - Un sistema distribuito, invece, è potenzialmente in grado di trattare con i malfunzionamenti, utilizzando (dinamicamente) componenti alternative per fornire funzionalità che alcune componenti non sono in grado temporaneamente di fornire

13

Requisiti non funzionali - 5

- Sistemi distribuiti *scalabili*
  - gestire i picchi di carico imprevedibili a cui possono essere soggetti
  - aumentare il throughput aggiungendo risorse senza modificare l'architettura
- Sistemi *sicuri*, per evitare che utenti non autorizzati possano accedere a dati sensibili
  - La sicurezza è ovviamente particolarmente complicata dalla natura remota dei sistemi distribuiti e della mobilità degli utenti, nodi e risorse al proprio interno
- Sistemi distribuiti che offrano *trasparenza*
  - mascherando dettagli e differenze del sistema sottostante

14

Organizzazione della lezione

- Un modello di riferimento per i Sistemi Distribuiti
  - Cosa è un modello di riferimento
  - Open Distributed Processing
- Le caratteristiche di un sistema distribuito
- I requisiti non funzionali di un sistema distribuito
- La trasparenza di un sistema distribuito
- Conclusioni

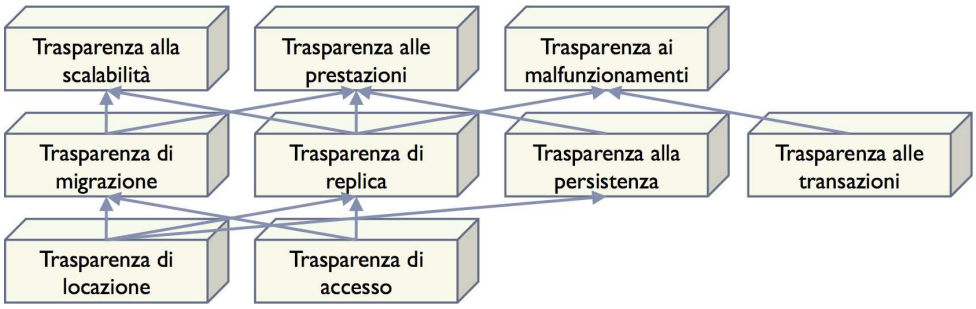
15

Un requisito non funzionale importante

- Caratterizza i Sistemi Distribuiti:
  - un Sistema Distribuito appare come una unica entità all'utente (utente finale, programmatore, progettista)
- I vantaggi della trasparenza
  - maggiore produttività (astrazione del modello)
  - alto riuso delle applicazioni sviluppate
- Diversi tipi di trasparenza, strettamente interconnessi su tre livelli:
  - **Trasparenza (livello di base):** di accesso e di locazione
  - **Trasparenza (livello di funzionalità):** di migrazione, di replica, di persistenza, di transazioni
  - **Trasparenza (livello di efficienza):** scalabilità, prestazioni, malfunzionamenti

16

La trasparenza: una visione d'insieme



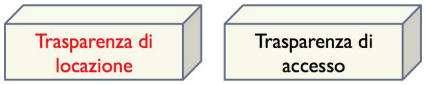
17

La trasparenza di ACCESSO

- Nasconde le differenze nella rappresentazione dei dati e nell'invocazione per l'interoperabilità tra oggetti
  - capacità di un sistema distribuito di presentare le risorse (come file, servizi, o dispositivi) agli utenti e alle applicazioni come se fossero locali, indipendentemente dalla loro effettiva posizione fisica nella rete
- Accesso ad oggetti attraverso la stessa interfaccia, sia da remoto che da locale
  - in questo modo un oggetto può essere facilmente spostato a run-time da un nodo ad un altro
- Fornito di default dai sistemi
  - trasparenza necessaria per garantire interoperabilità in un ambiente eterogeneo

18

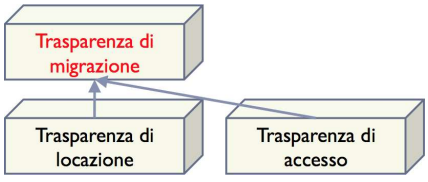
La trasparenza di LOCAZIONE



- Capacità di un sistema distribuito di nascondere agli utenti e alle applicazioni la posizione fisica delle risorse (come dati, servizi o dispositivi) all'interno della rete
- Non è permesso usare informazioni circa la posizione di una componente del sistema (localizzazione), componente usata in maniera indipendente dalla locazione
  - visione logica fornita dal sistema di naming
- Fondamentale per un sistema distribuito
  - fornito di default per rendere indipendenti dalla posizione i servizi da fruire

19

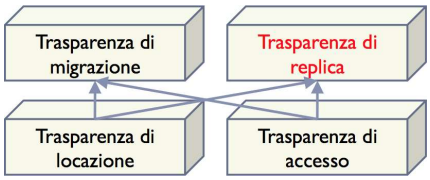
La trasparenza di MIGRAZIONE



- Il sistema può far migrare oggetti da un nodo del sistema ad un altro, senza che i fruitori dei suoi servizi ne siano a conoscenza
- Per ottimizzare prestazioni (bilanciamento carico) o per malfunzionamenti / riconfigurazioni
- Basata su trasparenza di accesso e locazione

20

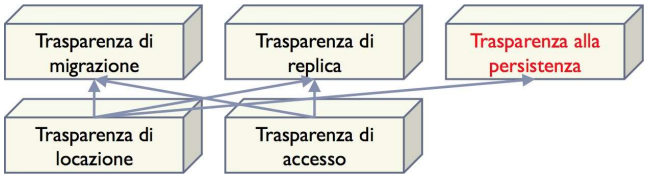
La trasparenza di REPLICA



- Un oggetto viene duplicato in copie (repliche) posizionate su altri nodi del sistema
  - il sistema si occupa di mantenere le repliche coerenti tra loro
- Usate per le prestazioni
  - porre i servizi laddove siano facilmente raggiungibili (come i Content Delivery Networks, tipo Akamai)
- Basata su trasparenza di accesso e locazione

21

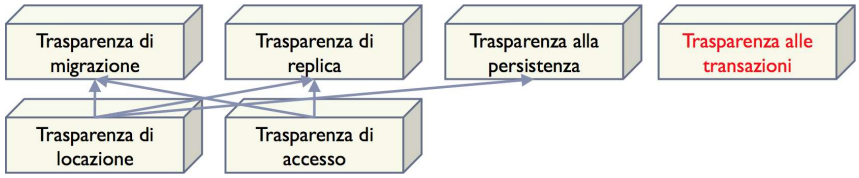
La trasparenza di PERSISTENZA



- L'oggetto è reso persistente (memoria secondaria) senza che l'utente se ne debba occupare
  - meccanismo di attivazione-deattivazione per risparmiare risorse su oggetti scarsamente utilizzati (handle)
- Basato su trasparenza di locazione:
  - oggetto re-attivato anche su nodi diversi da quelli dove è stato deattivato

22

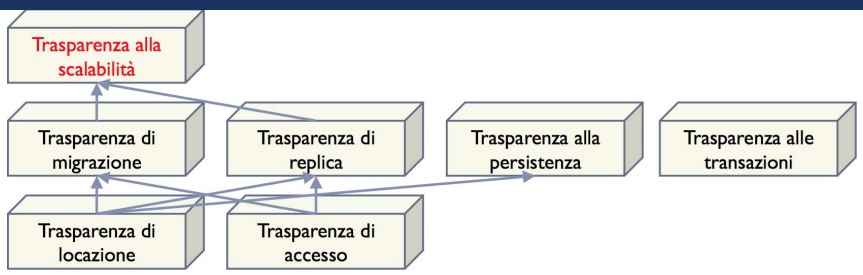
La trasparenza alle TRANSAZIONI



- Sistema implicitamente concorrente
- Transazioni garantite dal sistema
  - per offrire la coerenza del comportamento in presenza di accessi concorrenti
- Semplificazione notevole offerta dal sistema agli sviluppatori di applicazioni
- Si assicura che in caso di malfunzionamenti una risorsa non si trovi in uno stato non coerente

23

La trasparenza alla SCALABILITA'



- Un sistema è scalabile se in grado di poter servire carichi di lavoro crescenti senza dover modificare architettura ed organizzazione
  - aggiungendo ed integrando risorse
- Basata su migrazione e replica
  - nuove risorse verranno utilizzate, replicando i servizi che sono sotto carico, e facendoli migrare

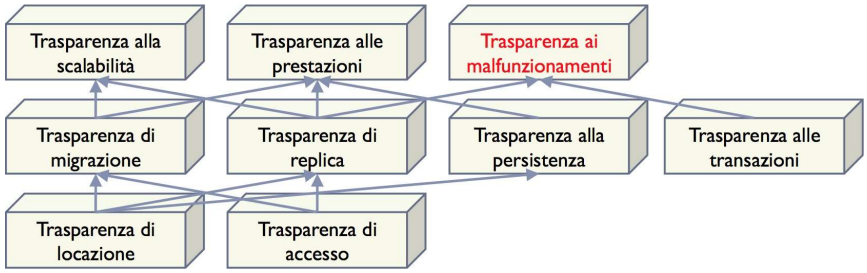
24

La trasparenza alle PRESTAZIONI

- Progettista/sviluppatori ottengono alte prestazioni dal sistema senza conoscerne i meccanismi utilizzati
  - bilanciamento del carico (migrazione/replica) minimizzazione della latenza (migrazione/replica) ottimizzazione risorse (persistenza)

25

La trasparenza ai MALFUNZIONAMENTI



- In presenza di malfunzionamenti di qualche componente, il resto del sistema riesce a fornire servizi (magari in maniera parziale)
- Basato su trasparenza di replica (non esistono componenti critiche) ma anche sulle transazioni (che permettono di fare il rollback di transazioni non complete, che vanno rieseguite su una replica)

26

Conclusioni

- Un modello di riferimento per i Sistemi Distribuiti
  - Cosa è un modello di riferimento
  - Open Distributed Processing
- Le caratteristiche di un sistema distribuito
- I requisiti non funzionali di un sistema distribuito
- La trasparenza di un sistema distribuito
- Conclusioni

27

Corso di Laurea in Informatica
III Anno Triennale
Programmazione Distribuita – classe 1

Middleware a oggetti distribuiti

Delfina Malandrino
dmalandrino@unisa.it
http://www.unisa.it/docenti/delfinamalandrino

28

Organizzazione della lezione

- 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

29

Organizzazione della lezione

- 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

30

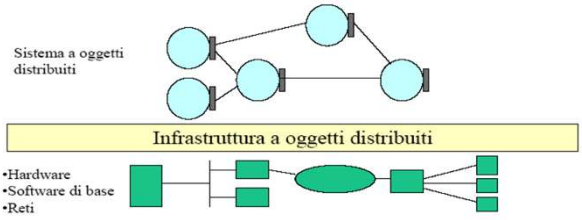
Gli oggetti distribuiti

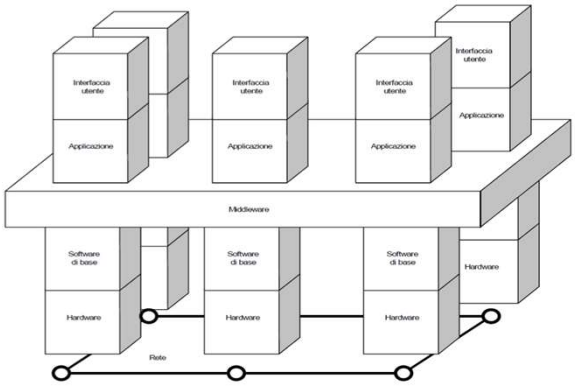
- Gli oggetti distribuiti si trovano alla confluenza di due aree della tecnologia software:
  - i *sistemi distribuiti*
    - che puntano a realizzare un unico sistema integrato basato sulle risorse offerte da diversi calcolatori messi in rete
  - lo *sviluppo e la programmazione orientata agli oggetti*
    - 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

31

Come si realizza questa integrazione?

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





32

A chi serve il middleware?

- 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

33

A chi serve il middleware?

- Scopo del middleware: rendere semplici tutti questi compiti, e fornire le astrazioni appropriate per i programmatori
  - che ben si integrino con gli strumenti tradizionali che usano per sviluppare l'applicazione

34

Organizzazione della lezione

- 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

35

Tipi di Middleware

- Middleware di **infrastruttura**
- Middleware di **distribuzione**
- Middleware **per servizi comuni**

36

Tipi di Middleware

- 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.

37

L'astrazione fornita dal Middleware

- 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

38

Organizzazione della lezione

- 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

39

Remote Procedure Call

- 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

40

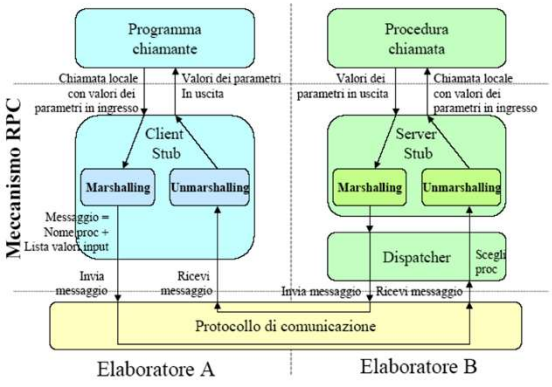
Le innovazioni di RPC

- 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

41

Le innovazioni di RPC

- 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)



42

42

Invocazioni sincrone

- Classico paradigma dell'invocazione di funzioni (metodi)
- La funzione chiamante (processo, thread, . . .) viene bloccata fino a quando la esecuzione dellq funzione (metodo) remota non ha terminato . . .
- . . . ed ha restituito il risultato
- Classica semantica di programmazione non-concorrente (familiare e di semplice utilizzo)
- **Invece … 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

43

Alcuni problemi e difficoltà di RPC

- 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

44

Organizzazione della lezione

- 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

45

Il passaggio da RPC a Oggetti Distribuiti

- 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…

46

Il passaggio da RPC a Oggetti Distribuiti

- … 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#, . . .)

47

Le motivazioni al passaggio

- 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

48

Organizzazione della lezione

- 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

49

Middleware CORBA

- 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 distrubuito
- 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

50

.NET Framework

- .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

51

Enterprise Java

- 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

52

Enterprise Java: componenti

- 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)

53

Organizzazione della lezione

- 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

54

Enterprise Java:
middleware implicito ed esplicito

- 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

55

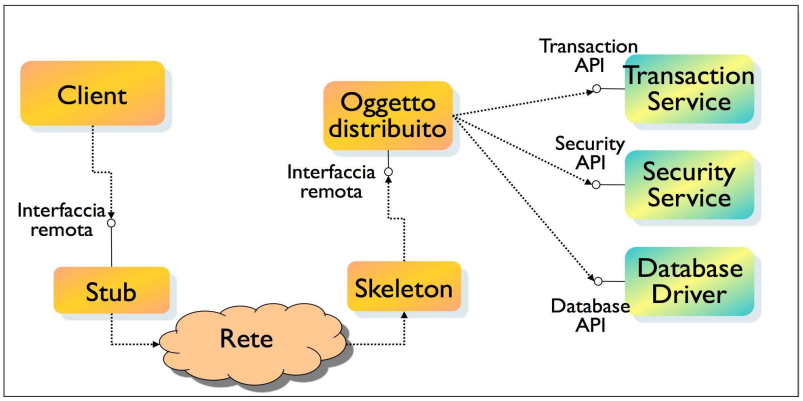
Enterprise Java:
middleware implicito ed esplicito

- Tra i servizi che devono essere messi a disposizione da un sistema a componenti, di particolare importanza sono quelli di fornire un protocollo di *comunicazione remota*, che permette le interazioni tra le componenti remote
- In questo ambito, le caratteristiche esibite dai Middleware ad Oggetti Distribuiti (come Java RMI) li hanno resi la base per il recente sviluppo dei sistemi a componenti
  - in quanto i meccanismi di comunicazione tra oggetti distribuiti permettono di offrire il supporto per le invocazioni di operazioni tra layer diversi di architetture software

56

Middleware esplicito

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



57

Un esempio di Middleware esplicito

```
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
}
```

Qual è la logica di business ?

58

Un esempio di Middleware esplicito

```
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

59

Un esempio di Middleware esplicito

```
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
}
```

Middleware

60

Un esempio di Middleware esplicito

```
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
}
```

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

61

Come vorremmo che fosse

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

Solo logica di business

62

Come vorremmo che fosse

```
//...
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

63

Come vorremmo che fosse

Obiettivo: usare il middleware senza conoscerne le API

```
//...
public void transfer(Account acct1, Account acct2, long amount) {

    //sottrai amount da account1 e somma amount in account2
}
```

Middleware implicito o dichiarativo

- codice dell'oggetto distribuito (senza MW)

64

Come vorremmo che fosse

Obiettivo: usare il middleware senza conoscerne le API

```
//...
public void transfer(Account acct1, Account acct2, long amount) {

    //sottrai amount da account1 e somma amount in account2
}
```

Middleware implicito o dichiarativo

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

65

Come vorremmo che fosse

Obiettivo: usare il middleware senza conoscerne le API

```
//...
public void transfer(Account acct1, Account acct2, long amount) {

    //sottrai amount da account1 e somma amount in account2
}
```

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!*"

66

Come vorremmo che fosse

```
//...
public void transfer(Account acct1, Account acct2, long amount) {

    //sottrai amount da account1 e somma amount in account2
}
```

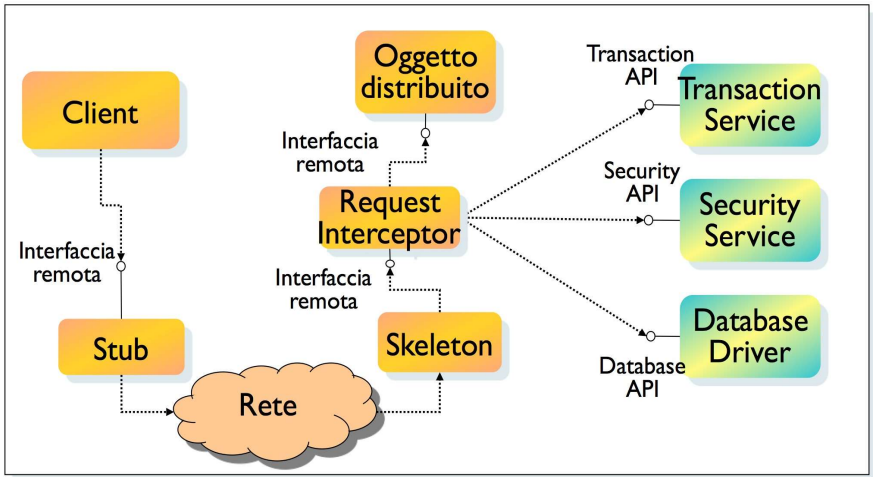
Obiettivo: usare il middleware senza conoscerne le API

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

67

Middleware implicito



68

Organizzazione della lezione

- 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

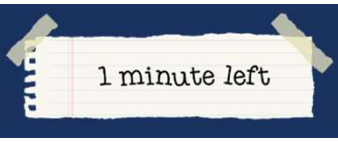
69

Il ruolo degli oggetti distribuiti… concludendo

- 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

70

Conclusioni



- 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

71

Corso di Laurea in Informatica
III Anno Triennale
Programmazione Distribuita

Middleware a oggetti distribuiti

Delfina Malandrino
dmalandrino@unisa.it
http://www.unisa.it/docenti/delfinamalandrino

72