

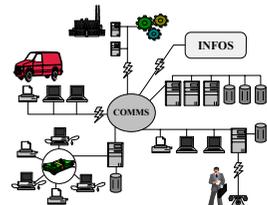
# Modelli ed Architetture



Web server



Web server



Private LAN



2001-2002

Modelli ed Architetture

1

## Struttura del corso

- **Standard di Riferimento, Architetture e Modelli**
  - **Modelli di Programmazione (Client/Server, Code Mobility)**
  - **Protocolli e Standard**
  - **Principi di Progettazione e Dimensionamento Sistemi**
- **Tecnologie e Linguaggi di Programmazione**
  - Il linguaggio HTML
  - Programmazione Client Side
  - Programmazione Server Side
  - Tecnologie e Standard nello sviluppo di Applicazioni Distribuite
  - Integrazione dei Sistemi Informativi con applicazioni Web-based
- **Tecnologie di Sviluppo avanzate basate sull'XML**
  - Il linguaggio XML
  - Il documento XML e la sua tipizzazione
  - Le tecnologie XML-based: XSL, XLL, XPOINTER
  - Il parser XML: i modelli DOM e SAX
  - Tecniche di sviluppo di Applicazioni Web-based basate sull'XML
  - Tecniche e Problematiche di System Integration

2001-2002

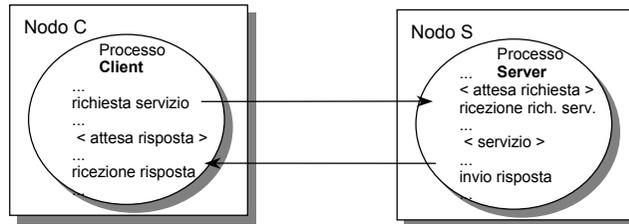
Modelli ed Architetture

2

## Il Modello Client Server

Il modello Client/Server prevede due entità:

- l'entità **Client** richiede il servizio
- l'entità **Server** offre il servizio



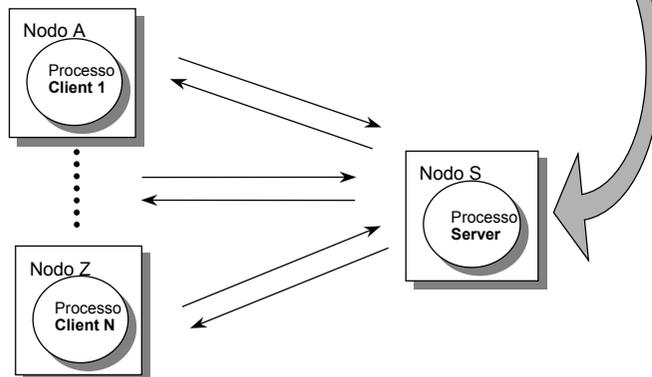
Il modello Client/Server risolve il problema del **rendez vous** (quando sincronizzare i processi comunicanti) definendo il **Server** come un processo **sempre in attesa** di richieste di servizio.

Si semplifica in questo modo il protocollo di comunicazione sottostante che non deve occuparsi di attivare un processo alla ricezione di un messaggio

## Il Modello Client Server

È un modello di comunicazione **asimmetrica, multi:1**

Il Cliente designa esplicitamente il destinatario  
Il Servitore risponde al processo che ha effettuato una richiesta



## Il Progetto del Client e del Server

Il Server deve accedere alle risorse del sistema:

- problemi di autenticazione utenti
- autorizzazione all'accesso
- integrità dei dati
- privacy delle informazioni

e deve gestire richieste contemporanee da molti Client (server concorrenti)



Maggiore complessità di progetto dei Server rispetto ai Client.

## Tipi di interazione tra Client e Server

Due tipi principali di interazioni:

- **interazione connection oriented**, viene stabilito un canale di comunicazione virtuale prima di iniziare lo scambio dei dati (es. connessione telefonica)
- **interazione connectionless**, non c'è connessione virtuale, ma semplice scambio di messaggi (es. il sistema postale)

La scelta tra i 2 tipi dipende dal tipo di applicazione e anche da altre caratteristiche proprie del livello di comunicazione sottostante.

Per esempio, in Internet il livello di trasporto è TCP oppure UDP:

- TCP è con connessione, inoltre è reliable (affidabile) e preserva l'ordine di invio dei messaggi
- UDP è senza connessione, non reliable e non preserva ordine messaggi

## Lo STATO dell'interazione tra Client e Server

L'interazione tra un Client e un Server può essere di due tipi:

- **stateful**, cioè esiste lo stato dell'interazione e quindi un messaggio dipende da quelli precedenti
- **stateless**, non si tiene traccia dello stato, ogni messaggio è indipendente dagli altri

Lo stato dell'interazione memorizzato nel Server (che quindi può essere stateless o stateful)

Un Server **stateful** ha migliore efficienza (dimensioni messaggi più contenute e migliore velocità di risposta del Server, presenta però problemi di replicazione)

Un Server **stateless** è più affidabile in presenza di malfunzionamenti (soprattutto causati dalla rete)

## Lo STATO dell'interazione tra Client e Server

La scelta tra server stateless o stateful deve tenere in conto anche (e soprattutto) le caratteristiche dell'applicazione.

Un'interazione stateless è possibile **SOLO** se il **protocollo** applicativo è progettato con **operazioni idempotenti**.

Operazioni idempotenti producono sempre lo stesso risultato, per esempio, un Server fornisce sempre la stessa risposta a un messaggio M indipendentemente dal numero di messaggi M ricevuti dal Server stesso.

Quando si ha un'interazione stateful il Server deve poter identificare il Client.

## Concorrenza nell'interazione tra Client e Server

### Lato Client

I Client sono programmi sequenziali, eventuali invocazioni concorrenti supportate dal sistema operativo multitasking.

### Lato Server

La concorrenza è cruciale per migliorare le prestazioni di un Server.

Un **Server iterativo** processa le richieste di servizio una alla volta.

Possibile basso utilizzo delle risorse, in quanto non c'è sovrapposizione tra elaborazione ed I/O.

Un **Server concorrente** gestisce molte richieste di servizio concorrentemente, cioè una richiesta può essere accettata anche prima del termine di quella (o quelle) attualmente in corso di servizio.

Migliori prestazioni ottenute da sovrapposizione elaborazione ed I/O.

La gestione di processi concorrenti implica una analisi precisa della sincronizzazione nell'accesso alle risorse (principi di atomicità delle transazioni e di consistenza dei dati)

## Quale tipo di Server? Riepilogo

		Tipo di comunicazione	
		con connessione	senza connessione
Tipo di Server	iterativo		
	concorrente		
	singolo processo		
	multi processo		

La scelta del tipo di Server dipende dalle caratteristiche del servizio da fornire

Vari tipi di Server, dipende dal **tipo di protocollo** e dalla **tecnologia** realizzativa.

Per esempio, in Unix è facile realizzare un Server concorrente generando un processo nuovo (fork) per ogni richiesta di servizio (server concorrente multi processo).

## Programmazione Client Side e Server Side

Questa classificazione deriva dal mondo Web; si definisce:

- **Esecuzione Server Side:** elaborazione effettuata dalla entità server, insieme delle operazioni che devono essere completate al fine di generare l'output per il Cliente
- **Esecuzione Client Side:** elaborazione effettuata dalla entità client, insieme delle operazioni necessarie per la gestione ed eventuale visualizzazione dei dati ottenuto dal server.

Nel mondo Web il contesto di esecuzione Client Side è il Browser il quale si occupa di interpretare i dati di output in formato html ottenuti dal server e visualizzarli graficamente.

La visualizzazione non è statica, l'interattività è ottenibile attraverso lo sviluppo di applicazioni Client side.

**Anche in Browser è un interprete**, è una macchina virtuale che mette in esecuzione le pagine web che ottiene a seguito di una request

## I Sistemi Distribuiti ed il Web

- **Applicazione Distribuita:**
  - Applicazione software realizzata attraverso la collaborazione di diverse entità in esecuzione su risorse computazionali fisicamente distinte.
- **Un Sistema Distribuito** è quindi un ambiente entro il quale possono essere operative una o più applicazioni distribuite.
- **Il mondo Web** da questo punto di vista è quindi un sistema distribuito:
  - Definisce un insieme di standard per la comunicazione (TCP/IP, HTTP/HTML ...)
  - Fornisce un insieme di servizi di supporto (DNS, NIC, Certification Authority ...)

## A cosa serve una Applicazione Distribuita?

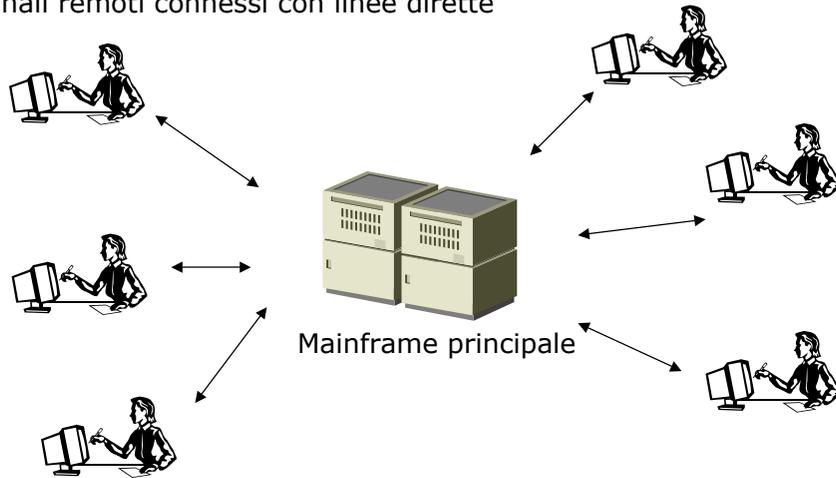
- **Condivisione delle Risorse:**
  - **Risorse Dati** (Database con le più diverse informazioni e tipologia e struttura di accesso)
  - **Risorse computazionali** (capacità di calcolo, memoria a breve e lungo termine)
  - **Risorse per l'accesso a periferiche e canali di comunicazione** (fax, posta elettronica, sms, comunicazione vocale (voice over IP), stampanti o altre periferiche...)
- **Applicazione principi di economia di scala**

## Vantaggi nelle Applicazioni distribuite: Sistema di biglietteria

- **Specifiche:**
  - Fornire disponibilità posti in tempo reale
  - Permettere l'acquisto dei biglietti da diversi punti vendita dislocati in luoghi fisicamente separati
  - Alto volume di prenotazioni (es. aerei o treni)
- **Necessità:**
  - Fornitura di un elevato livello di servizio:
    - Interfacce rapide ed efficienti
    - Garanzia di continuità di servizio

## Sistema Biglietteria Soluzione Monolitica

Terminali remoti connessi con linee dirette

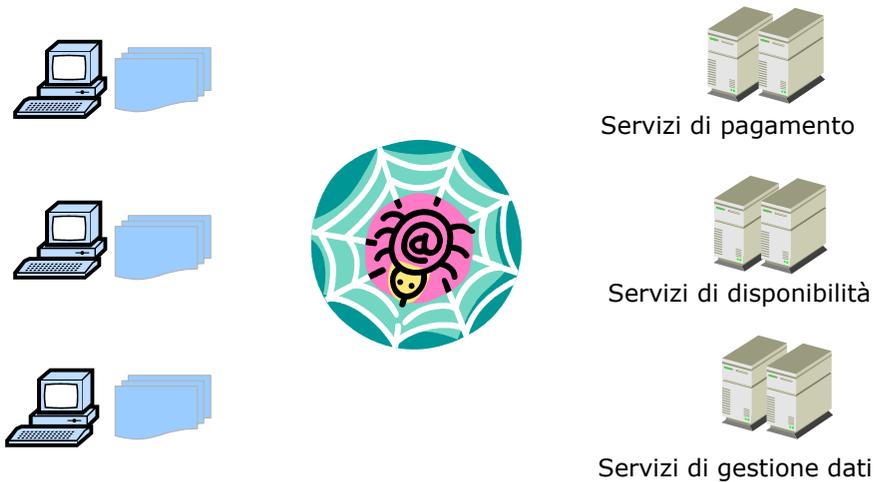


2001-2002

Modelli ed Architetture

15

## Sistema Biglietteria Soluzione Distribuita



2001-2002

Modelli ed Architetture

16

## Sistema Biglietteria Vantaggi e Svantaggi delle due Soluzioni

### Soluzione Monolitica

#### Vantaggi:

- Sistema facile da implementare
- Una sola macchina (mainframe) che contiene tutte le informazioni

#### Svantaggi:

- Regge un numero definito di utenti
- Le interfacce sono molto semplici e spartane
- Un guasto nel server centrale provoca una perdita di servizio

### Soluzione Distribuita

#### Vantaggi:

- è possibile affiancare diversi server in parallelo per aumentare le prestazioni e la tolleranza ai guasti
- I client possono presentare le informazioni ottenute dai server in modo indipendente, con grafica e strumenti per la semplificazione del lavoro.
- è possibile agganciare un numero di Client proporzionale alle prestazioni (scalabilità)

#### Svantaggi:

- Ci sono client specializzati (che devono essere installati e devono essere compatibili con i diversi HW a disposizione)
- La manutenzione implica la necessità di effettuare update di software decentralizzati presso i Client
- La realizzazione di un ambiente distribuito è più complessa di un ambiente monolitico

## Sistema Biglietteria Soluzione Distribuita Web-based



Web Server



Servizi di disponibilità



Servizi di pagamento



Servizi di gestione dati

## Sistema Biglietteria Soluzione Distribuita Web-based

### Vantaggi:

- **Tutti i vantaggi evidenziati dai sistemi distribuiti in generale**
- interfaccia tra Client e Server standardizzata permette di creare applicazioni senza la necessità di installare un Client dedicato sui Client
- Il Web Browser diventa un Client general purpose utile per realizzare le applicazioni più diverse
- La standardizzazione dello sviluppo implica una semplificazione nella realizzazione

### Svantaggi:

- Il modello di interazione Client-Server è predefinito e non permette interattività forte
- L'interfaccia utente è limitata alle funzioni che lo standard definisce, impoverendosi rispetto alle prestazioni di un Client dedicato

## InterNet – IntraNet - ExtraNet

- **InterNet:**
  - rete di accesso pubblico per la diffusione di applicazioni distribuite... (esistono mille definizioni a seconda della moda)
- **IntraNet:**
  - rete aziendale o riservata basata sulle stesse tecnologie di InterNet all'interno della quale operano applicazioni web-based il cui accesso è strettamente riservato all'azienda o ente.
- **ExtraNet:**
  - insieme di applicazioni web-based fruibili via InterNet con accesso riservato a determinati utenti per usi specifici.
  - rappresenta l'estensione dei Sistemi IntraNet sulla rete pubblica per particolari esigenze.

## Sistemi Distribuiti Sistemi Web-based

- Un sistema Web-based è un sistema distribuito.
- Soddisfa tutti i requisiti e le specifiche di un sistema distribuito
- Si basa su standard e tecnologie che consentono di soddisfare tali requisiti
  - Protocolli di comunicazione
  - Modelli e Tecnologie applicative (client side e server side)
  - Architetture e strutture implementative

## Sistemi Distribuiti Specifiche e Requisiti

**condivisione delle risorse**

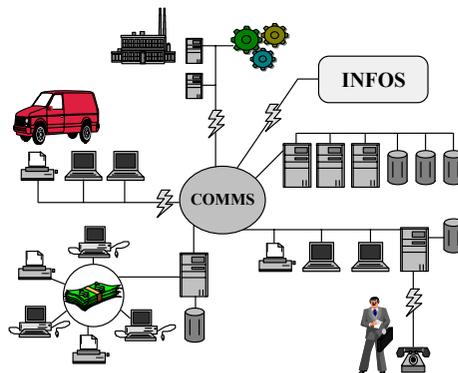
**affidabilità** per tollerare guasti

**applicazioni intrinsecamente distribuite**

(es. prenotazioni aeree)

**Accessibilità** del sistema

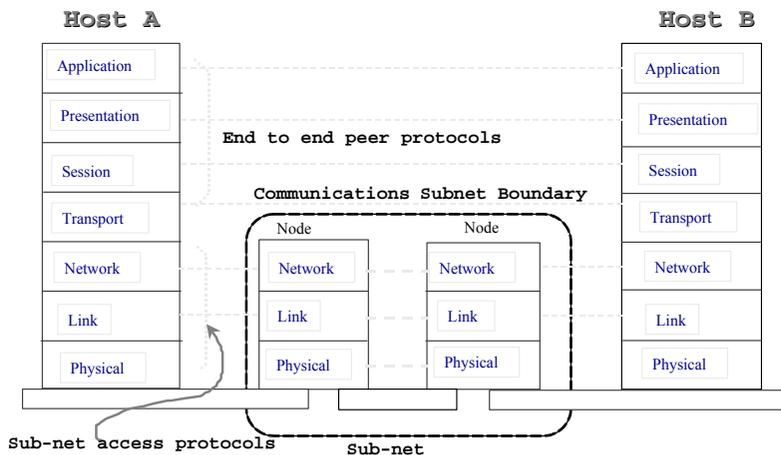
**scalabilità**



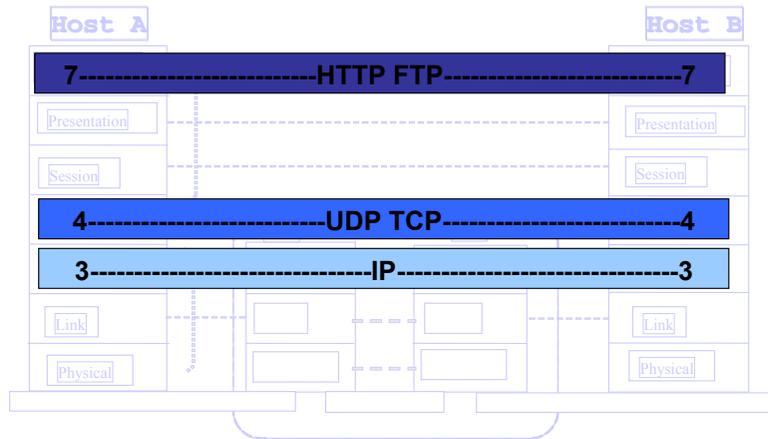
## Sistemi Distribuiti Come soddisfare i Requisiti ?

- **Apertura:**
  - **Definizione di Standard** (TCP/IP, HTTP, FTP, HTML... ma anche CORBA, Java...)
- **Concorrenza:**
  - Dimensionamento e progettazione architetture in grado di soddisfare esigenze di accesso concorrente
- **Trasparenza:**
  - Necessità di realizzare Servizi che siano in grado di nascondere la complessità della implementazione
- **Scalabilità:**
  - **Teorica** (impostazione architeturale)
  - **Pratica** (applicazione reale della architettura)
- **Tolleranza ai Guasti**
  - **Robustezza**
  - **Availability**
  - **Reliability**

## Protocolli e Standard ISO - OSI



## Protocolli e Standard TCP/IP, HTTP, FTP ...



2001-2002

Modelli ed Architetture

25

## HTTP: Hyper Text Transfer Protocol

- Basato sul modello Client/Server
- Il Client effettua una richiesta HTTP:
  - Necessità di una **Risorsa**
  - Invia un **Messaggio** di richiesta della **Risorsa**
- Il Server invia la risposta:
  - Riceve il **Messaggio** di richiesta della **Risorsa**
  - Esegue le elaborazioni necessarie a fornire la risposta
  - Invia un **Messaggio** di risposta (serve la **Risorsa** richiesta)
- Gli elementi in gioco nel protocollo sono:
  - Il **Messaggio** HTTP (richiesta e risposta)
  - La **Risorsa**

2001-2002

Modelli ed Architetture

26

## HTTP: Hyper Text Transfer Protocol Terminologia

- **Client:** Programma applicativo che stabilisce una Connessione al fine di inviare delle *Request*
- **Server:** Programma applicativo che accetta Connessioni al fine di ricevere *Request* ed inviare specifiche *Response*
- **Connessione:** circuito virtuale stabilito a livello di trasporto tra due applicazioni per fini di comunicazione
- **Messaggio:** è l'unità base di comunicazione HTTP, è definita come una specifica sequenza di byte concettualmente atomica.
  - **Request:** messaggio HTTP di richiesta
  - **Response:** messaggio HTTP di risposta
  - **Resource:** Oggetto di tipo dato univocamente definito
    - **URI:** Uniform Resource Identifier – identificatore unico per una risorsa.
- **Entity:** Rappresentazione di una Risorsa, può essere incapsulata in un messaggio.

## HTTP: Hyper Text Transfer Protocol Messaggio

Un messaggio HTTP è definito da due strutture:

- **Message Header:** Contiene tutte le informazioni necessarie per la identificazione del messaggio.
- **Message Body:** Contiene i dati trasportati dal messaggio.

Esistono degli schemi precisi per ogni tipo di messaggio relativamente agli header ed ai body.

I messaggi di Response contengono i dati relativi alle risorse richieste (nel caso più semplice la pagina html)

I dati sono codificati secondo il formato specificato nell'header, solitamente sono in formato MIME; è possibile utilizzare anche il formato ZIP

## HTTP: Hyper Text Transfer Protocol URL

- **Uniform Resource Locator:** rappresenta l'estensione dell'**URI** tenendo conto del protocollo necessario per il trasferimento della risorsa. Per il protocollo HTTP l'URL è il seguente:

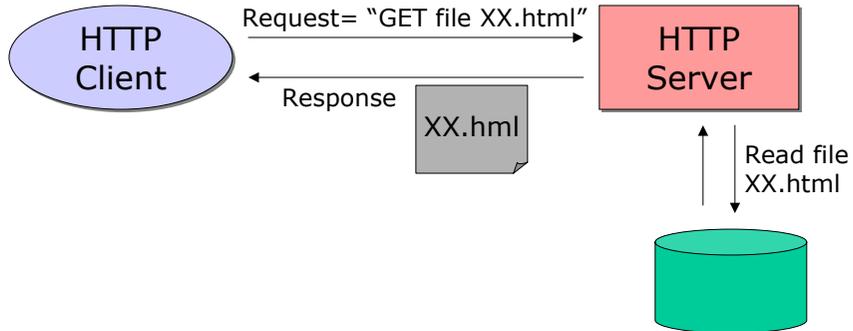
**http\_URL = "http:" "://" host [ ":" port ] [ abs\_path [ "?" query ] ]**

- Se la porta non viene specificata viene scelta la **porta 80** come da default dello standard
- Se il path non viene specificato interviene il percorso di root del Web Server
- La chiave **"?"** serve per la specifica degli eventuali parametri nella richiesta della risorsa (*chiamata in **get***)

## HTTP: Hyper Text Transfer Protocol Terminologia

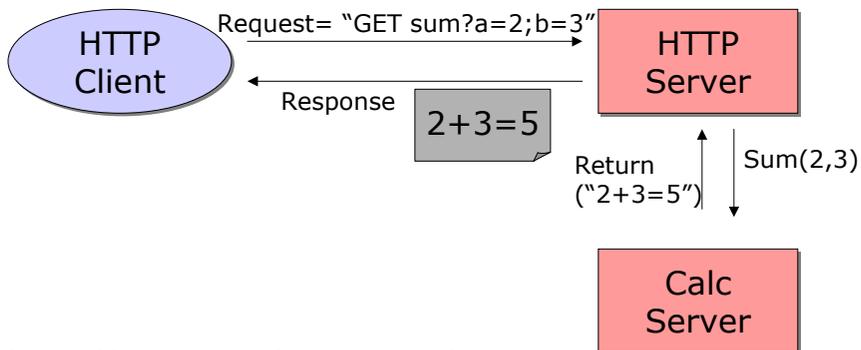
- **Proxy:** Programma applicativo in grado di agire sia come Client che come Server al fine di effettuare richieste per conto di altri Clienti. Le *Request* vengono processate internamente oppure vengono ridirezionate al Server. Un proxy deve interpretare e, se necessario, riscrivere le *Request* prima di inoltrarle.
- **Gateway:** Server che agisce da intermediario per altri Server. Al contrario dei proxy, il gateway riceve le request come se fosse il server originale ed il Client non è in grado di identificare che la Response proviene da un gateway.
- **Cache:** Repository locale di messaggi di *Response*, compreso il sottosistema che controlla la consistenza dei dati. Qualsiasi Client o Server (tranne i tunnel) può includere una cache per motivi di performance. Un *Response* si dice **Cacheable** se il contenuto è memorizzabile senza perdita di consistenza.

## HTTP: Hyper Text Transfer Protocol Request/Response 1



`http://myserver/XX.html`

## HTTP: Hyper Text Transfer Protocol Request/Response 2



`http://myserver/sum?a=2;b=3`

## HTTP: Hyper Text Transfer Protocol Metodi di Request

- **GET:** richiedo una specifica risorsa attraverso un singolo URL, posso passare diversi parametri, **la lunghezza massima di un URL è 256 caratteri**
- **POST:** richiedo una specifica risorsa evidenziando che seguirà un altro messaggio la cui entity contiene i dettagli per la identificazione della risorsa stessa: **non ci sono limiti di lunghezza nei parametri di una richiesta**

Ci sono anche altri metodi, che permettono di verificare la versione di una risorsa, la compatibilità del server, le caratteristiche delle risorse...

Il metodo GET e POST vengono gestiti trasparentemente dal server HTTP, questo significa che dal punto di vista dello sviluppatore è trasparente se la richiesta è avvenuta tramite un metodo o l'altro. Quando si può scegliere, è sempre preferibile il metodo POST che non pone limiti di lunghezza massima dei parametri.

## HTTP: Hyper Text Transfer Protocol I Cookie

Parallelamente alle sequenze request/response, il protocollo prevede una struttura dati che si muove come un token, dal client al server e vice versa: **i Cookie**

**I cookie** sono in realtà una tupla di stringhe:

- **Key:** identifica univocamente un cookie all'interno di un dominio:path
- **Value:** valore associato al cookie (è una stringa di max 255 caratteri)
- **Path:** posizione nell'albero di un sito al quale è associato (di default /)
- **Domain:** dominio dove è stato generato
- **Max-age:** (opzionale) numero di secondi di vita (permette la scadenza di una sessione)
- **Secure:** (opzionale) non molto usato prevede una verifica di correttezza da parte del server
- **Version:** identifica la versione del protocollo di gestione dei cookie (1)

I cookie possono essere generati sia dal client che dal server, dopo la loro creazione vengono sempre passati ad ogni trasmissione di request e response.

## Modelli ed Architetture

- I **modelli** sono la sintesi, lo scheletro teorico all'interno del quale è possibile costruire applicazioni reali
- Ci sono diversi modelli, applicabili a diversi contesti, con i loro limiti ed i loro vantaggi
- Le **architetture** dettano la composizione delle risorse a disposizione al fine di ottenere strutture anche eterogenee in grado di assolvere a compiti complessi
- Le configurazioni architettureali devono avere specifiche caratteristiche di flessibilità, semplicità e scalabilità al fine di permettere un corretto sviluppo del sistema

## Modelli e Tecnologie

- **Paradigmi di Programmazione:**
  - **Strutturata** (ANSI C, Visual Basic, Perl, linguaggi di scripting... )
  - **Object Oriented** (C++, Java...)
  - **Dichiarativa** (Prolog, XSL...)
- **Modelli di Coordinamento:**
  - **Client/Server**
  - **Agenti mobili**
- **Modelli di computazione distribuita:**
  - **CORBA**
  - **COM/DCOM**
  - **EJB**

## Modelli e Tecnologie

### Linguaggi "normali" e Linguaggi di Scripting

- Il linguaggi "normali" sono i linguaggi general purpose, come C o Java.
- Un linguaggio si dice di scripting quando i programmi non devono rispettare delle strutture precise ma l'esecuzione viene "descritta" dai singoli comandi
- I linguaggi di scripting sono sempre eseguiti da un **interprete**, sono caratterizzati da set estesissimi di funzioni standard che permettono di realizzare applicazioni anche complesse in pochissime linee.
- Sono strettamente dipendenti dal contesto di utilizzo:
  - **Shell di unix**: linguaggio di scripting focalizzato essenzialmente sulla manipolazione dei file e dei processi
  - **Tcl/Tk**: linguaggio una volta molto diffuso per la realizzazione di interfacce testuali
  - **Perl**: nato come linguaggio di shell si è evoluto come linguaggio "general purpose" per la manipolazione dei dati; molto popolare per la realizzazione di CGI
  - **Javascript**: semplificazione di Java nato per la gestione del sistema ad eventi dei Browser si è affermato come linguaggio di riferimento
  - **Vbscript** (Visual Basic Script): controparte di Microsoft del Javascript

2001-2002

Modelli ed Architetture

37

## Modelli e Tecnologie

### Paradigmi di Coordinamento: Client/Server

Il modello Client/Server è un modello di coordinamento, evidenzia le modalità di interazione tra entità attive

- Modello di base su cui si fonda qualsiasi interazione
- Ogni configurazione anche complessa può essere ricondotta ad uno schema Client/Server
- Per realizzare una applicazione di rete:
  - Identificazione delle identità che devono collaborare
  - Definizione dei Ruoli: Clienti e Servitori
  - Sviluppo del protocollo di interazione tra Clienti e Servitori

2001-2002

Modelli ed Architetture

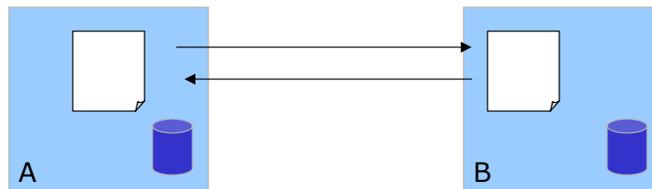
38

## Modelli e Tecnologie

### Paradigmi di Coordinamento: Client/Server

Il modello Client/Server in configurazioni particolarmente complesse crea delle strutture molto difficili da sviluppare e da gestire

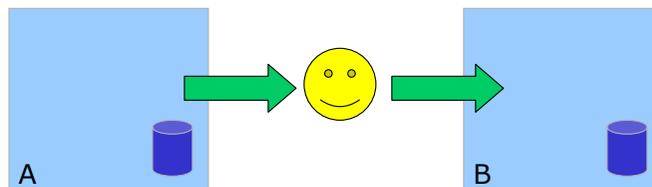
La divisione in entità è dettata non solo dalla logica del servizio, ma dal concetto di "località". Se devo intraprendere una azione che richiede due risorse dislocate su due nodi remoti sono costretto a dividere l'azione tra due entità, una che effettua la elaborazione su un nodo (A) e chiede all'altra l'elaborazione sull'altro nodo (B)



## Modelli e Tecnologie

### Paradigmi di Coordinamento: Gli Agenti Mobili

E se l'entità potesse *muoversi* da un nodo all'altro per accedere alle diverse risorse ?



Si parlerebbe in questo caso di **Agente** perché non sarebbe classificabile né come Cliente né come Servitore

Non sarebbe più necessario lo sdoppiamento in due entità distinte per la sola necessità di località alla risorsa

Questi Agenti si dicono infatti **Mobili** perché sono in grado di svincolarsi dal limite della località. Questo non significa che non viene applicato più il modello cliente servitore, viene semplicemente utilizzato al di sopra della struttura di rete, non più vincolato dalla struttura fisica sottostante

## Modelli e Tecnologie

### Modelli di Computazione Distribuita

#### **CORBA è uno standard aperto, garantisce:**

- Language neutral:
- Platform independent:
- Versatilità, eterogeneità e flessibilità

La sua estrema ampiezza costa però in termini di **Complessità realizzativa**; non sono previste infatti astrazioni forti di ausilio allo sviluppo (Gestione Transazioni, Sicurezza, Replicazione).

#### **Le tecnologie più diffuse per la computazione distribuita sono:**

- **DCOM**: tecnologia proprietaria Microsoft basata sul modello a Componenti, nata dalla evoluzione del concetto di **DLL**:
  - Al di fuori degli standard della computazione distribuita dettati da CORBA
  - Realizzabile solo attraverso tool di sviluppo microsoft
  - Dipendenza dalla piattaforma WinXX
- **EJB** Enterprise Java Bean: è l'implementazione della interfaccia CORBA in Java, per rendere aperte le applicazioni java-based
  - Java based
  - Platform independent:
  - Versatile e potente
  - Semplicità realizzativa: implementa i cosiddetti "Distributed Services", grazie ai quali è possibile astrarre lo sviluppo

## Modelli e Tecnologie

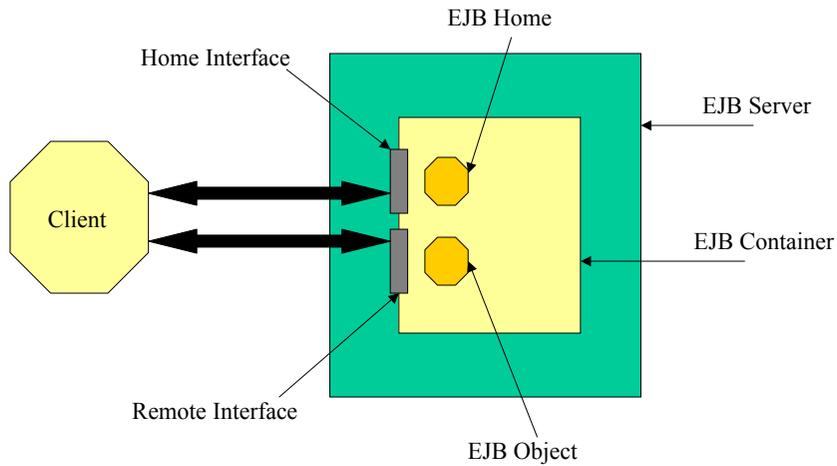
### EJB – Distributed Services

Un EJB è un oggetto Java che viene implementato all'interno di una struttura predefinita e la cui esecuzione viene gestita da un motore (Container). Una entità computazionale può accedere ai servizi dell'EJB attraverso il modello RMI; in questo senso l'entità viene definita genericamente Client.

Con EJB tutto ciò che riguarda la gestione dei servizi è disponibile ed utilizzabile in maniera del tutto trasparente al Client:

- **Transazioni**: Il Container EJB offre un sistema automatico di gestione delle transazioni dati
- **Security**: eredita il modello di sicurezza di Java
- **Scalabilità**: la tecnologia implementa policy di clustering e load balancing

## Modelli e Tecnologie Schema d'uso di un EJB



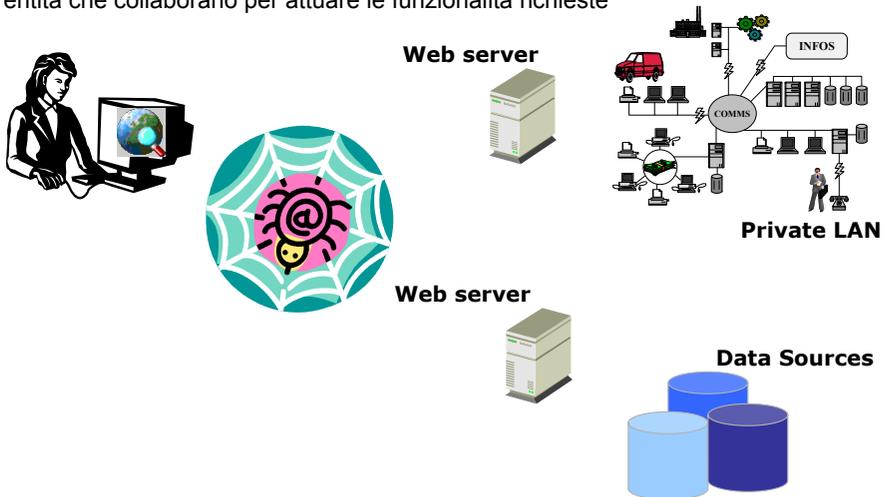
2001-2002

Modelli ed Architetture

43

## Architetture dei Sistemi Web

L'**Architettura** di un Sistema Distribuito Web-based è l'organizzazione di un insieme di entità che collaborano per attuare le funzionalità richieste



2001-2002

Modelli ed Architetture

44

## Architetture dei Sistemi Web

È possibile pensare di realizzare una architettura uniforme che permetta di sviluppare applicazioni Web distribuite in grado di lavorare al di sopra delle tecnologie standard offerte.

Quali devono essere le **specifiche principali** di questa architettura ?

- Completa aderenza allo standard HTTP
- Completa compatibilità con i Browser Web disponibili
- Apertura alle diverse tecniche e tecnologie di sviluppo software
- *Ingegnerezabilità* del software e del processo di sviluppo
- Scalabilità
- Tolleranza ai Guasti

## Architetture dei Sistemi Web Specifiche

### **Completa aderenza allo standard HTTP**

Ipotesi fondamentale per garantire la completa trasparenza di tutte le infrastrutture di rete alle applicazioni.

Questa ipotesi permette di usufruire di un Browser standard, normali linee di rete basate sulla suite TCP/IP, al di sopra di strutture di rete eterogenee, in piena compatibilità con i servizi di DNS, security, monitoring intrinseci della infrastruttura.

## Architetture dei Sistemi Web Specifiche

### **Completa compatibilità con i Browser Web disponibili**

I Browser Web diventano una “scatola” dove eseguire le interfacce delle applicazioni.

Queste “scatole” sono però abbastanza delicate, sono realizzate infatti da un insieme di interpreti che elaborano i dati che provengono dai diversi server

Per garantire questa specifica è necessario attuare diversi accorgimenti qualitativi al fine che il codice realizzato sia compatibile con i diversi Browser in commercio.

## Architetture dei Sistemi Web Specifiche

### **Apertura alle tecniche e tecnologie di sviluppo software**

Il successo di una architettura è spesso vincolato alla apertura della stessa alla evoluzione tecnologica.

La realizzazione di una struttura indipendente dalla tecnologia di sviluppo permette di sviluppare con strumenti diversi, adeguando di volta in volta la struttura alle esigenze

## Architetture dei Sistemi Web Specifiche

### **Ingegnerizzabilità del software e del processo di sviluppo**

La creazione di applicazioni complesse evidenzia la necessità di applicare tecniche di ingegneria del software in particolare per garantire le seguenti proprietà:

- Previsione e pianificazione dei tempi di lavoro
- Parallelizzazione e Specializzazione nello sviluppo dei diversi componenti del software
- Accelerazione dei tempi di realizzazione
- Semplicità di manutenzione ed evoluzione del codice una volta realizzato

## Architetture dei Sistemi Web Specifiche

### **Scalabilità**

La scalabilità è una proprietà tipica dei sistemi distribuiti, nei sistemi Web diventa fondamentale per la realizzazione di applicazioni in grado di servire diversi target di Utenti in volumi anche molto diversi.

Questa proprietà può essere risolta a livello architetturale o applicativo; le soluzioni architetturali offrono dei servizi standard, trasparenti allo sviluppo, le soluzioni applicative permettono a volte di garantire performance importanti

## Architetture dei Sistemi Web Specifiche

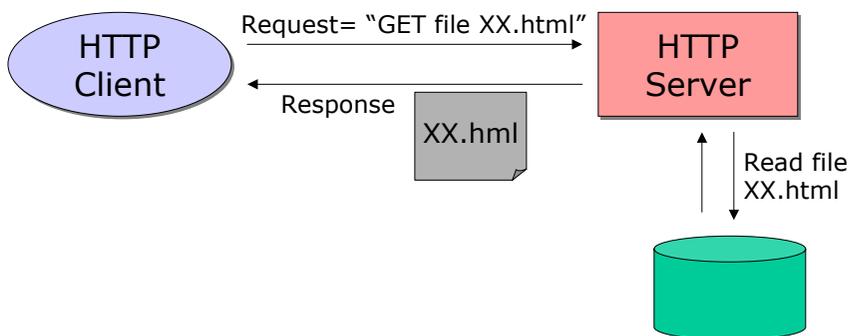
### Tolleranza ai Guasti

Nella realizzazione di servizi online, è necessario garantire dei livelli di servizio. Questi livelli di servizio sono ottenibili solo attraverso l'applicazione di architetture che permettono la replicazione dei servizi.

La replicazione può essere realizzata secondo due logiche:

- **Replicazione a Risorse Fredde:** se rimane attiva una sola istanza di servizio e sono disponibili una o più risorse in attesa di sostituire il servizio attivo in modo trasparente
- **Replicazione a Risorse Calde:** si può replicare i servizi mantenendo attive tutte le istanze a disposizione, questo permette di sfruttare tutte le risorse attraverso di politiche di distribuzione e bilanciamento del carico

## Architetture dei Sistemi Web Modello di Base

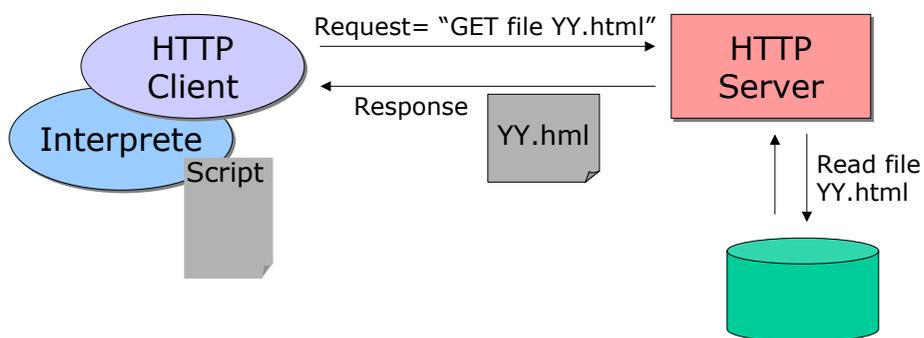


<http://myserver/XX.html>

## Architetture dei Sistemi Web Modello di Base – Analisi Specifiche

- **Completa aderenza allo standard HTTP:** OK
- **Completa compatibilità con i Browser Web disponibili:** dipende dalla qualità con cui vengono scritte le pagine html, esistono tools di sviluppo che permettono di verificare ed ottimizzare la *compatibilità cross-browser* anche nello sviluppo di pagine in dhtml.
- **Apertura alle diverse tecniche e tecnologie di sviluppo software:** nessuna apertura, le applicazioni Web così fatte offrono solo un ambiente di navigazione ad ipertesti per la consultazione dei dati contenuti, non è possibile sviluppare codice al di fuori dell'html stesso
- **Ingegnerizzabilità del software e del processo di sviluppo:** molto limitata, possibilità di applicare principi di riusabilità del codice in semplici contesti ma con scarsi risultati
- **Scalabilità:** ottima, il server è stateless, posso affiancare quanti server desidero che insistano sugli stessi sources, posso replicare sia i server che i sources.
- **Tolleranza ai Guasti:** ottima, la ottengo implicitamente dalla possibilità di replicare il servizio

## Architetture dei Sistemi Web Modello di Base con Programmazione Client side



<http://myserver/YY.html>

## Architetture dei Sistemi Web

### Modello di Base con Programmazione Client side

### Analisi Specifiche

- **Completa aderenza allo standard HTTP:** OK
- **Completa compatibilità con i Browser Web disponibili:** dipende dalla qualità con cui vengono scritte le pagine html e dalla capacità del Browser di interpretare il linguaggio di scripting.
- **Apertura alle diverse tecniche e tecnologie di sviluppo software:** minima, limitata alle capacità dell'interprete del Browser
- **Ingegnerizzabilità del software e del processo di sviluppo:** limitata, possibilità di applicare principi di riusabilità del codice realizzando semplici librerie importabili in diverse pagine
- **Scalabilità:** ottima, il server è stateless, posso affiancare quanti server desidero che insistano sugli stessi sources, posso replicare sia i server che i sources.
- **Tolleranza ai Guasti:** ottima, la ottengo implicitamente dalla possibilità di replicare il servizio

2001-2002

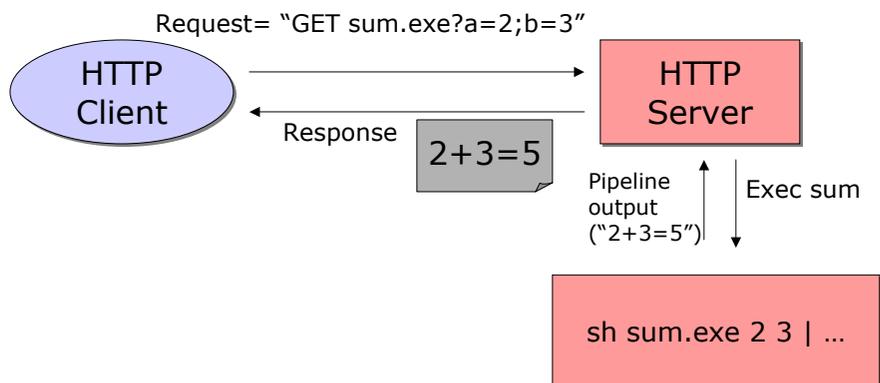
Modelli ed Architetture

55

## Architetture dei Sistemi Web

### CGI

- **CGI – Common Gateway Interface:** Il Web Server passa le chiamate alle applicazioni realizzate secondo una logica simile ad un "filtro unix"



<http://myserver/sum.exe?a=2;b=3>

2001-2002

Modelli ed Architetture

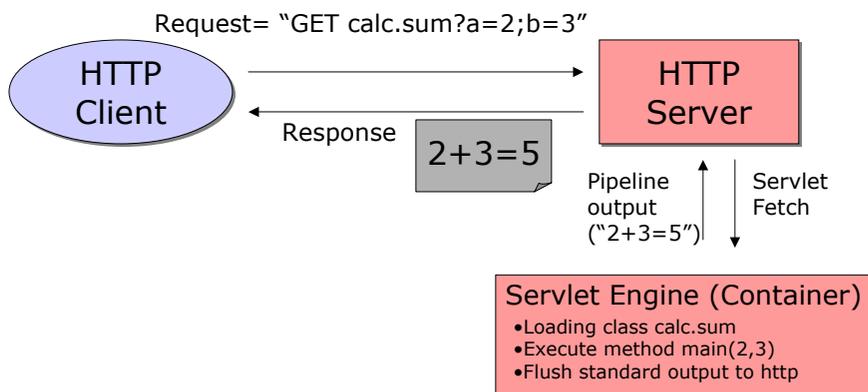
56

## Architetture dei Sistemi Web CGI – Analisi Specifiche

- **Completa aderenza allo standard HTTP:** OK
- **Completa compatibilità con i Browser Web disponibili:** dipende *solamente* dalla qualità con cui vengono scritti i programmi CGI, non c'è nessun supporto a livello architetturale che garantisca la qualità dell'output.
- **Apertura alle diverse tecniche e tecnologie di sviluppo software:** è possibile realizzare programmi CGI con diverse tecniche e linguaggi; sono molto usati linguaggi sia scripting come il perl o tcl/tk ma è possibile utilizzare anche il C.
- **Ingegnerizzabilità del software e del processo di sviluppo:** la struttura delle applicazioni è molto frammentata, spesso si opta per una rapida prototipizzazione attraverso linguaggi di scripting piuttosto che cercare di applicare tecniche di ingegneria del software su strutture realizzate da enormi quantità di filtri disaggregati.
- **Scalabilità:** ottima, anche in questo caso il server è stateless, posso affiancare quanti server desidero che insistano sugli stessi sources, posso replicare sia i server che i sources.
- **Tolleranza ai Guasti:** ottima, la ottengo implicitamente dalla possibilità di replicare il servizio. Data la tipologia del server, MTBF è molto più elevato relativamente ad un server statico

## Architetture dei Sistemi Web Java Servlet

- **Un Servlet** è una classe Java in grado di ricevere in modo strutturato i parametri e generare, sullo Standard Output, la pagina html di risposta



<http://myserver/calc.sum?a=2;b=3>

## Architetture dei Sistemi Web Java Servlet – Analisi Specifiche

- **Completa aderenza allo standard HTTP:** OK
- **Completa compatibilità con i Browser Web disponibili:** dipende dalla qualità con cui vengono scritte le servlet, non esiste nessun controllo di correttezza del codice html di output
- **Apertura alle diverse tecniche e tecnologie di sviluppo software:** questa architettura si basa strettamente sulla tecnologia Java, l'apertura viene quindi concettualmente demandata a livello di linguaggio. Java viene definito come un linguaggio aperto ed adatto all'interoperabilità ed alla integrazione in sistemi aperti
- **Ingegnerizzabilità del software e del processo di sviluppo:** anche in questo caso questa proprietà risente delle caratteristiche di Java. È possibile programmare secondo un modello ad oggetti sia attivi che passivi. La struttura della servlet rende abbastanza complessa la creazione della pagina Web di output. Non è possibile separare la logica di elaborazione dalla grafica vera e propria
- **Scalabilità:** il Servlet Engine è stateful, è possibile mantenere oggetti attivi in memoria che interagiscono con le servlet. La replicazione dipende quindi o dalla implementazione di strutture di clustering da parte del servlet engine oppure da una soluzione a livello applicativo.
- **Tolleranza ai Guasti:** segue di pari passo le problematiche di replicazione evidenziate per la scalabilità

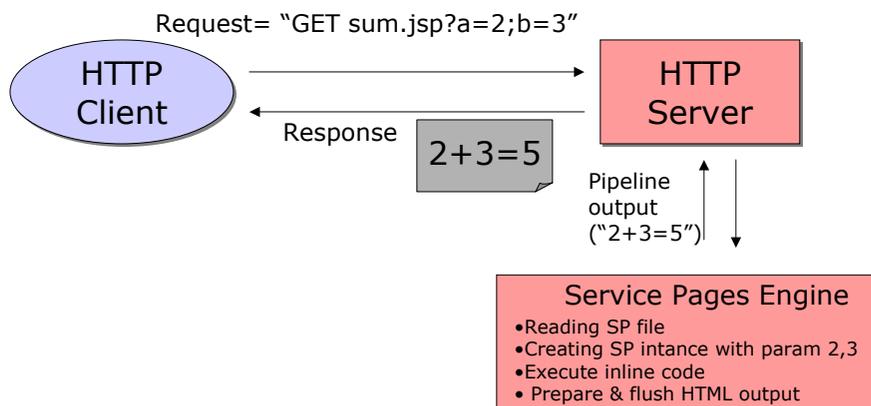
2001-2002

Modelli ed Architetture

59

## Architetture dei Sistemi Web Service Pages

- **Una Service Page** è una pagina html che contiene al suo interno del codice che deve essere eseguito sul lato server prima che la pagina venga inviata al client



<http://myserver/sum.jsp?a=2;b=3>

2001-2002

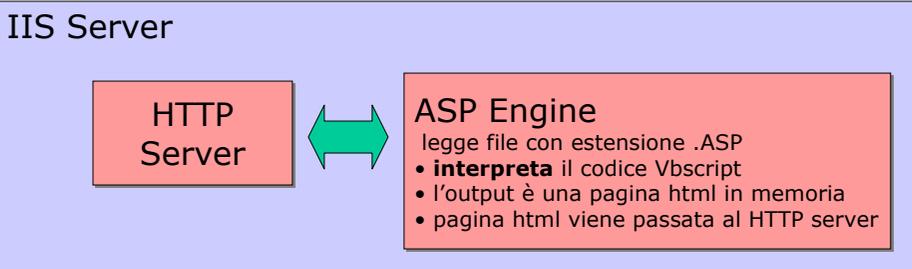
Modelli ed Architetture

60

## Architetture dei Sistemi Web Service Pages – Microsoft ASP

La architettura a *Service Pages* è stata implementata per la prima volta da Microsoft come soluzione di rapida prototipizzazione delle pagine Web dinamiche utilizzando Microsoft Internet Information Server. La soluzione Microsoft è nota come **ASP-Active Service Pages**, è stato il primo strumento per lo sviluppo di applicazioni Web-based che non richiedesse conoscenze specifiche di programmazione di rete.

Il linguaggio utilizzato per la realizzazione delle pagine ASP è il **VBScript**, una forma semplificata del Visual Basic di Microsoft.



2001-2002

Modelli ed Architetture

61

## Architetture dei Sistemi Web Service Pages -JSP

La struttura di esecuzione delle Java Servlet, grazie all'estrema flessibilità del linguaggio, è compatibile con la architettura a service pages. Si parla in questo caso di **JSP- Java Service Pages**.

Concettualmente, ASP e JSP sono la stessa cosa, con la sola differenza del linguaggio di scrittura delle SP.

Dal punto di vista architetturale invece la struttura JSP è significativamente diversa:

- Esiste un preCompilatore Java (JSP compiler): Il JSP compiler converte la pagina JSP in una servlet che poi viene compilata come una classe java standard
- Il linguaggio utilizzato all'interno delle pagine JSP è puro Java e non, come nel caso delle pagine ASP da un sottoinsieme. In particolare in JSP si possono istanziare altre classi Java, in ASP non è possibile invocare direttamente funzioni esterne (si possono istanziare solo oggetti COM)

Queste differenze hanno importanti ripercussioni sulla estensione della architettura a Service Page nelle due tecnologie

2001-2002

Modelli ed Architetture

62

## Architetture dei Sistemi Web Service Pages – Analisi Specifiche

- **Completa aderenza allo standard HTTP:** OK
- **Completa compatibilità con i Browser Web disponibili:** esistono dei tools che aiutano lo sviluppo di pagine SP secondo determinate specifiche qualitative
- **Apertura alle diverse tecniche e tecnologie di sviluppo software:** dipende dalla implementazione: ASP è una tecnologia proprietaria quindi completamente chiusa; per le JSP valgono tutte le considerazioni fatte per le Java servlet
- **Ingegnerizzabilità del software e del processo di sviluppo:** è un modello che permette un rapido sviluppo di applicazioni ma, preso da solo, non offre nessun strumento di ingegnerizzazione del software
- **Scalabilità:** come il Servlet Engine, anche gli SP engine sono in generale stateful, le diverse tecnologie offrono diverse soluzioni al problema della scalabilità
- **Tolleranza ai Guasti:** segue di pari passo le problematiche di replicazione evidenziate per la scalabilità

## Modelli e Tecnologie Sessioni e Conversazioni

**La Sessione** rappresenta lo stato associato ad una sequenza di pagine visualizzate da un utente:

- Contiene tutte le informazioni necessarie durante l'esecuzione; possono essere *informazioni di sistema* (ip di provenienza, lista delle pagine visualizzate...) oppure *informazioni di natura applicativa* (nome e cognome, username, quanti e quali prodotti ha inserito nel basket per un acquisto...)
- È l'unico stato necessario per la gestione di una applicazione web

**La Conversazione** rappresenta una sequenza di pagine di senso compiuto, (ad esempio l'insieme delle pagine necessarie per comperare un prodotto)

- È univocamente definita dall'insieme delle pagine che la compongono e dall'insieme delle interfacce di input/output per la comunicazione tra le pagine (detto **flusso della conversazione**)

## Modelli e Tecnologie Una Conversazione di Acquisto Online



1. Inizia la Conversazione, l'utente inserisce username e password: Il server ottiene i dati e li verifica con i dati presenti nel database dei registrati: **viene creata la sessione:**

- Username
- Nome e Cognome
- ...



2. Utente è autorizzato, sfoglia il catalogo alla ricerca di un prodotto; il server lo riconosce attraverso i dati di sessione



3. Trova il prodotto e lo mette nel carrello: **la sua sessione viene aggiornata con le informazioni del prodotto**

4. Compila i dati di consegna



2001-2002



5. Provvede al pagamento, **fine della conversazione di acquisto**

Modelli ed Architetture

A questo punto l'ordine viene memorizzato nella base dati e viene successivamente inviato ai sistemi logistici per la spedizione.  
• La sessione è ancora attiva e l'utente può fare un altro acquisto o uscire dal sito

65

## Modelli e Tecnologie Gestione della Sessione

- La Sessione ha quindi i seguenti requisiti:
  - Deve essere condivisa dal Client e dal Server
  - È associata ad una o più conversazioni effettuate da un singolo utente
  - Ogni utente possiede la sua singola sessione
- Ci sono due tecniche di base per gestire la sessione:
  1. Utilizzo della struttura dei **cookie**
  2. Gestione di uno stato sul server per ogni utente collegato
- I cookie fanno parte dello standard http, sono quindi sempre disponibili
- La gestione dello stato sul server è possibile in alcune architetture specifiche

2001-2002

Modelli ed Architetture

66

## Architetture dei Sistemi Web Struttura Multi-livello delle Applicazioni

Sviluppare applicazioni secondo una logica ad oggetti o a componenti significa scomporre l'applicazione in blocchi, servizi e funzioni.

È molto utile separare logicamente le funzioni necessarie in una struttura multi-livello al fine di fornire astrazioni via via più complesse e potenti a partire dalle funzionalità più elementari

Nel tempo si è affermata una classificazione indipendente dalla implementazione tecnologica, basata su una struttura a 4 livelli principali.

Questa struttura, non fornisce dettagli implementativi, non specifica quali moduli debbano essere implementati client side o server side, ne nessuna altra specifica tecnica: **è una architettura essenzialmente logico funzionale**

## Architetture dei Sistemi Web Struttura Multi-livello delle Applicazioni

### Presentation Level

Il livello di presentazione si occupa della visualizzazione dei dati generati dalle logiche secondo il percorso definito nel flusso

### Business Flows

A questo livello vengono implementati i flussi delle diverse conversazioni che compongono una applicazione

### Business Logic

Le Logiche di Business contengono le caratteristiche delle applicazioni in termini di modello dei dati e di logiche di utilizzo degli stessi.

### Services

I Servizi devono fornire tutte le funzionalità base necessarie, dalle interfacce di accesso ai sistemi informativi di supporto alla gestione dei sistemi di mailing o specifiche periferiche. I Servizi implementano le API di base per lo sviluppo delle applicazioni

## Architetture dei Sistemi Web Services

Realizzano le funzioni di base per lo sviluppo di applicazioni:

- **Accesso e gestione delle sorgenti di dati:**
  - Database locali
  - Sistemi informativi remoti
  - Legacy Systems (termine generico che identifica applicazioni esterne per la gestione aziendale – dal mainframe ad altri sistemi web-based)
- **Accesso e gestione risorse:**
  - Stampanti
  - sistemi fax, sms, email
  - Dispositivi specifici, macchine automatiche...
- **Sistemi di gestione e monitoraggio della applicazione**
  - Gestione della sessione web
  - Gestione degli errori applicativi
  - Logging e tracing della applicazione

## Architetture dei Sistemi Web Business Logic

La logica è l'insieme di tutte le funzioni ed i servizi che l'applicazione offre; questi servizi si basano sulle strutture di basso livello dei Services per implementare i diversi algoritmi di risoluzione e provvedere alla generazione dei dati di output.

Esempi di moduli di business logic possono essere:

- Gestione delle liste di utenti:
- Gestione cataloghi online

A questo livello, non è significativa quali sono le sorgenti di dati (nascoste dal livello di services) come non è significativo come arrivano le richieste di esecuzione dei servizi e come vengono gestiti i risultati ai livelli superiori

I moduli di business logic (siano essi implementati in componenti che in oggetti) mantengono il contenuto funzionale (la cosiddetta logica di business) che può essere utilizzata in diversi contesti, non vincolati a determinate interfacce o conversazioni

## Architetture dei Sistemi Web Business Flow

Una conversazione è realizzata da un insieme di pagine collegate in un flusso di successive chiamate.

Il business flow raccoglie quindi l'insieme delle chiamate necessarie per realizzare una conversazione; ogni chiamata deve caricare i parametri in ingresso, chiamare le funzioni di business logic necessarie per effettuare l'elaborazione e generare l'output che dovrà essere visualizzato.

Un flusso identifica quindi univocamente una conversazione, l'astrazione della business logic permette di studiare l'esecuzione delle singole pagine in modo indipendente dalla struttura dei dati e degli algoritmi sottostanti

## Architetture dei Sistemi Web Presentation Level

Il business flow è in grado di fornire i dati di output necessari; il **livello di presentazione** ha il compito di interpretare questi dati e generare l'interfaccia grafica per la visualizzazione dei contenuti

Questi due livelli sono concettualmente divisi poiché la generazione dei dati è logicamente separata dalla sua rappresentazione e formattazione.

Questo permette di avere diverse tipi di rappresentazione degli stessi dati, per esempio una rappresentazione in italiano ed una in inglese o una in html ed una in plain ASCII

## Architetture dei Sistemi Web Realizzazione di Architetture Multi-livello

Non tutte le tecnologie permettono di rispettare questa suddivisione, in molti casi i sistemi vengono realizzati a 2 o 3 livelli.

Queste semplificazioni portano in certi casi a miglioramenti nelle performance ma comportano un netta riduzione della leggibilità e della rapidità di sviluppo. Come sempre il trade off viene deciso in base al contesto, non esiste la soluzione ideale ad ogni situazione

Esempio di semplificazione a 2 livelli:

**Presentation Level    Business Flows**

---

**Business Logic        Services**

Una struttura a 2 livelli può essere realizzata fondendo i due livelli più bassi: Blogic e Services ed i due più alti Presentation e Bflows.

Il livello più basso ora non è più indipendente dalle sorgenti di dati, non posso riutilizzare le logiche su diverse basi dati ad esempio.

Il livello più alto invece riunisce la struttura delle conversazioni con la loro rappresentazione, diventa quindi impossibile modificare la formattazione in modo indipendente dalla conversazione e vice versa

## Architetture dei Sistemi Web Struttura di un Framework per l'eCommerce

### Presentation Level



### Business Flows



### Business Logic



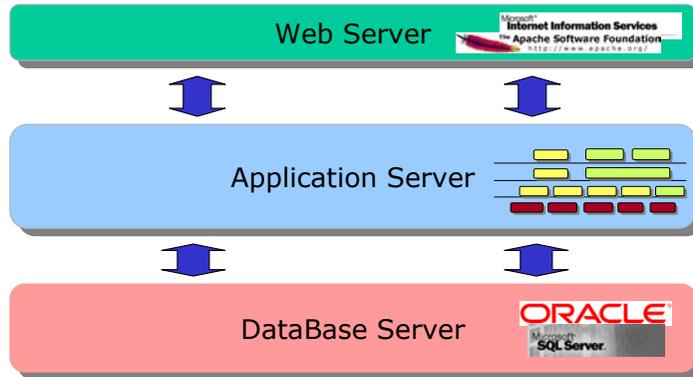
### Services



## Architetture dei Sistemi Web

### Architettura SW/HW e divisione dei servizi

La architettura applicativa, nella sua separazione logica in layer, rappresenta il razionale con cui viene sviluppato il cosiddetto layer applicativo di un Sistema Web:



## Architetture dei Sistemi Web

### Distribuzione dei Servizi

La struttura a 3 livelli rispecchia i 3 principali servizi che realizzano un sistema Web.

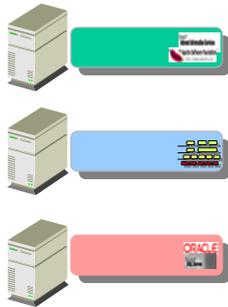
Questi 3 servizi possono risiedere sullo stesso HW oppure essere divisi su tre macchine separate:

Si parla in questo caso di **Distribuzione verticale** della architettura; è molto efficiente perché non necessita di nessun accorgimento specifico, viene realizzata essenzialmente per motivi di performance soprattutto quando si dividono il livello applicativo da quello database. Questo tipo di distribuzione non prevede replicazione, non è quindi utile per risolvere problemi di fault tolerance.

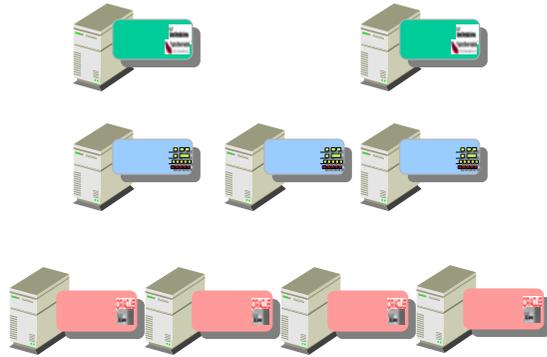
Orizzontalmente ad ogni livello è possibile replicare il servizio su diverse macchine; si parla in questo caso di **Distribuzione orizzontale**, necessita di importanti accorgimenti strettamente dipendenti dalla tecnologia d'uso. Essendo una distribuzione per replicazione è possibile implementare politiche per la gestione della fault tolerance

## Architetture dei Sistemi Web Configurazioni Distribuite e Replicate

### Distribuzione Verticale



### Distribuzione Orizzontale



2001-2002

Modelli ed Architetture

77

## Architetture dei Sistemi Web Replicazione DataBase – I Cluster

Il database server è un server stateful; la replicazione è molto delicata perché deve mantenere il principio di atomicità delle transazioni.

I database commerciali, come Oracle e Microsoft SQL Server prevedono delle configurazioni di clustering in grado di gestire in modo trasparente un numero variabile di CPU e macchine distinte.

Ci sono diverse configurazioni, a risorse calde e fredde, la performance è ottenibile solo attraverso repliche a risorse calde, la tolleranza a guasti viene sempre implementata attraverso l'uso di sistemi raid per i dati e di replicazione a coppie delle unità di database

2001-2002

Modelli ed Architetture

78

## Architetture dei Sistemi Web Replicazione Applicazione

L'unico stato necessario a livello applicativo è caratterizzato dalla sessione.

È possibile che il servizio di applicazione utilizzi oggetti o componenti con stato per motivi di performance (caching) o necessità specifiche.

Alcuni framework disponibili sul mercato permettono la replicazione attraverso tecniche di clustering molto simili a quelle dei sistemi database, altri framework non sono in grado di replicare orizzontalmente.

Se si mantiene lo stato concentrato all'interno della sessione, e la sessione viene gestita attraverso i cookie, è possibile realizzare il framework applicativo completamente stateless, ottenendo una configurazione completamente replicabile orizzontalmente.

## Architetture dei Sistemi Web Replicazione Web Server

Il web server è stateless, non crea problemi nella replicazione.

L'unicità degli URL può essere gestita attraverso diverse soluzioni sia hardware che software.

Si possono applicare politiche di load balancing e load sharing con diverse euristiche

