




Framework


Impianti Informatici

POLITECNICO DI MILANO





Web application - tecnologie



Web Application: tecnologie

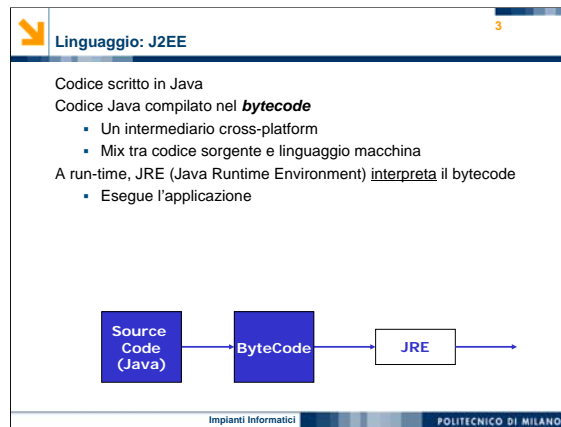
2

- Java-based (**J2EE**)
 - Sviluppata inizialmente da Sun
 - Cross-platform e open source
 - Gestire direttamente le funzionalità dell'applicazione
 - Benefici della comunità di utenti
- Microsoft-based (**.Net**)
 - Componenti proprietarie
 - Tecnologia limitata alle piattaforme Microsoft
 - Teoricamente accetta molteplici linguaggi di programmazione

Impianti Informatici POLITECNICO DI MILANO

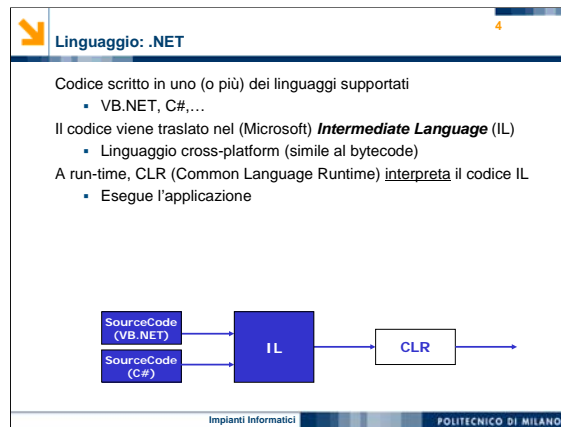
Attualmente esistono due tecnologie leader per la realizzazione di web application: quella J2EE e quella .NET

1. Quella Java è stata sviluppata
2. inizialmente da SUN; è basata su un linguaggio
3. Cross-platform e Open Source, quindi, oltre ad essere gratuito ed utilizzabile su qualsiasi macchina abbia installata una Java Virtual Machine, permette spesso di disporre dei codici sorgenti e quindi
4. gestire in maniera più diretta il funzionamento dell'applicazione.. ma soprattutto consente di godere dei
5. benefici della comunità, quindi del sw sviluppato da altri, così come dei consigli degli altri programmatori...
6. Quella Microsoft utilizza
7. componenti proprietarie, di cui spesso si dispone solo dei codici binari, quindi non modificabili e adattabili a piacimento a seconda delle necessità.
8. Lo svantaggio principale della tecnologia .NET è di essere limitata alle sole piattaforme microsoft.
9. Tra gli aspetti più propositivi del framework, emerge la propensione ad essere language-independent e la possibilità di far interoperare fra loro linguaggi differenti. Questo permette di integrare applicativi scritti in linguaggi ormai obsoleti.



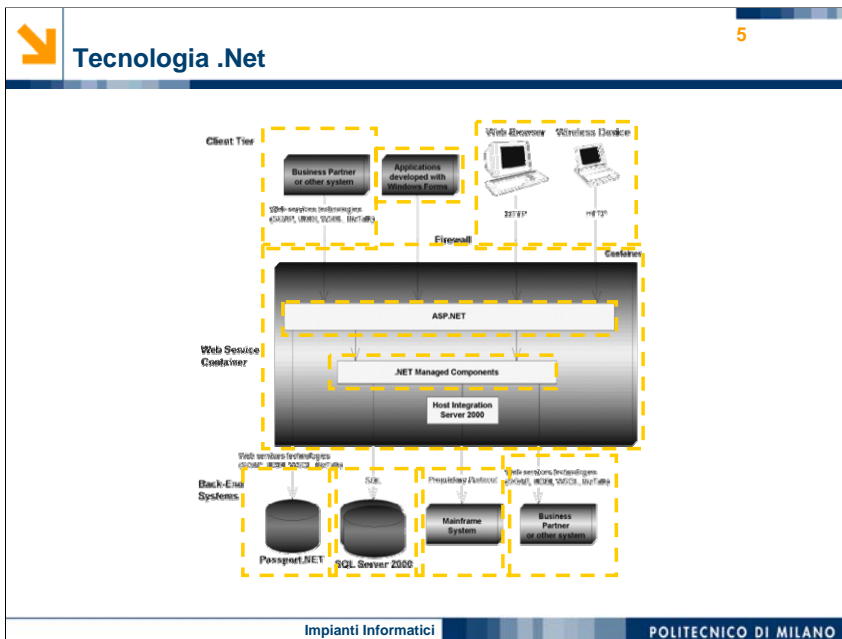
La tecnologia J2EE è basata sul linguaggio di programmazione JAVA, col vantaggio di poter essere eseguito in sistemi eterogenei.

1. Il codice Java, infatti, viene
2. compilato nel cosiddetto bytecode, un linguaggio
3. cross-platform, intermediario tra
4. il codice sorgente e il linguaggio macchina.
5. A runtime, Java Runtime Environment interpreta il bytecode
6. e lo esegue



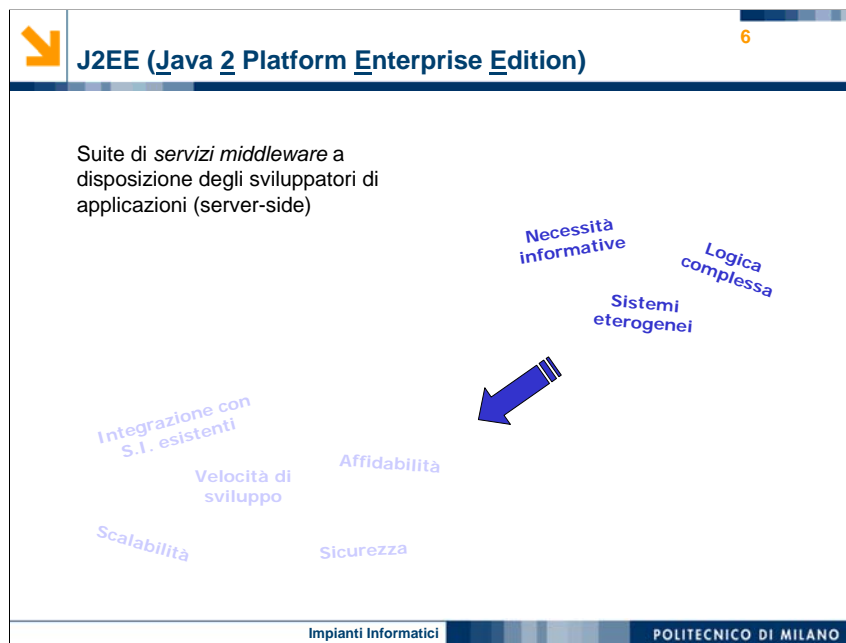
Nella tecnologia .NET il linguaggio di programmazione di riferimento è C#, anche se è teoricamente svincolata dall'uso un particolare linguaggio.

1. Il codice scritto in uno dei linguaggi supportati, anche parzialmente in più linguaggi,
2. come VB.NET o lo stesso C#,
3. viene traslato nel Microsoft Intermediate Language, un
4. linguaggio cross-platform nello stile del bytecode java.
5. A run time, Common Language Runtime interpreta l'Intermediate Language
6. ed esegue l'applicazione



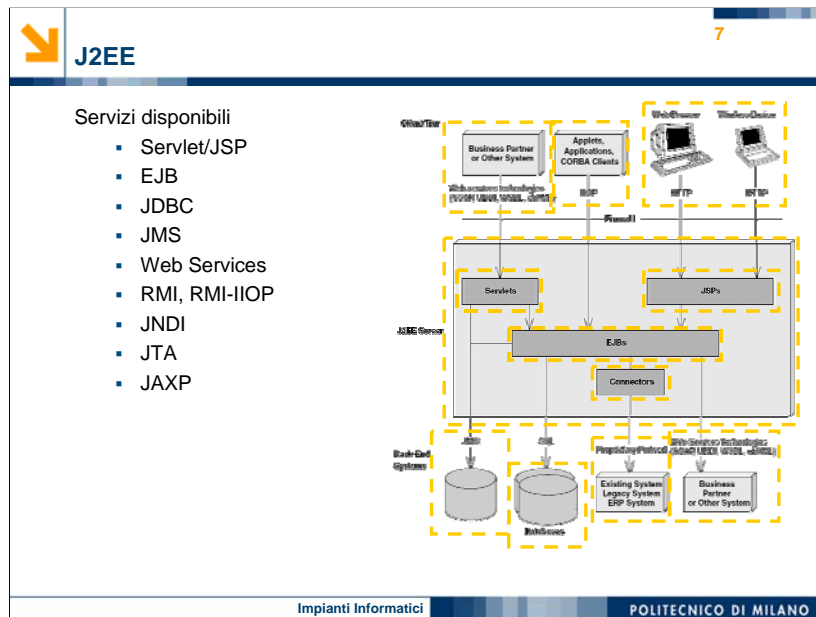
Le applicazioni .NET sono supportate

1. da un container, che fornisce la quality of service richiesta dall'applicativo, come transazioni, sicurezza e servizi di messaging.
2. Il business layer delle applicazioni .NET, che si occupa del processamento dei dati e della logica applicativa, viene realizzato utilizzando i cosiddetti Managed Component
3. è connesso ai database attraverso l'ADO.NET, ovvero l'Active Data Objects, e ai sistemi esistenti usando altri servizi,
4. quali COM TI, cioè il COM Transaction Integrator. Può inoltre essere connesso ad altri
5. partner business mediante tecnologia web services;
6. gli stessi partner business possono essere connessi all'applicazione .NET tramite web services.
7. I client fat, i browser web e i device wireless sono connessi a pagine ASP.NET, le Active Server Pages, che si occupano di effettuare il rendering della pagina da presentare all'utente;
8. interfacce utenti complesse possono essere costruite usando le Windows Form.



Vediamo ora con maggiore dettaglio alcune componenti della tecnologia J2EE, attualmente la più diffusa tra le due piattaforme di sviluppo, per il fatto di essere open source.

1. J2EE rappresenta un framework per lo sviluppo di applicazioni complesse lato server. Il software enterprise ha infatti:
2. peculiari necessità informative,
3. logiche particolarmente complesse,
4. necessità di essere eseguito su sistemi eterogenei.
5. J2EE mette a disposizione dei servizi che permettono di
6. velocizzare lo sviluppo di applicazioni, rimanendo aggiornati con gli ultimi standard.
Facilitano inoltre la realizzazione di applicazioni
7. Affidabili,
8. Sicure,
9. Scalabili, consentendo un'agevole integrazione
10. coi sistemi informativi esistenti



La figura mostra l'integrazione delle diverse tecnologie offerte dalla piattaforma J2EE.

1. Il core del framework sono i vari *container*, inclusi in uno o più application server j2ee-compliant.
2. Essi gestiscono componenti quali
3. Le servlet, le pagine jsp
4. e gli Enterprise Java Bean.
5. La comunicazione con i database è supportata dalla tecnologia JDBC, java database connectivity, usando il linguaggio sql.
La comunicazione con altri sistemi di back-end può essere gestita, ad esempio, mediante
6. JMS, oppure con
7. protocolli proprietari, o ancora mediante
8. la tecnologia dei web services.
La chiamata di oggetti remoti può avvenire tramite
9. RMI, o RMI-IIOP.
10. Web browser e device wireless si connettono tramite il protocollo http a pagine JSP.
I diversi componenti sono inoltre supportati da tecnologie quali
11. JNDI, la Java Naming and Directory Interface, che fornisce un meccanismo standard per la localizzazione di risorse, come oggetti remoti o servizi di directory.
12. JTA, acronimo di Java Transaction Api per la gestione di transazioni
13. JAXP, Java Api for Xml Parsing, che offre supporto per il parsing di file XML



Un middleware ad oggetti consente la

1. comunicazione tra due oggetti distribuiti, come due differenti oggetti in esecuzione su due macchine differenti.
2. È basato sui concetti della Remote Procedure Call, inventata all'inizio degli anni 80 per chiamare una procedura su una macchina differente da quella su cui si sta lavorando.
3. Un middleware ad oggetti consente di chiamare non solo procedure, ma anche oggetti remoti.
4. Le funzionalità che deve offrire un middleware rendono trasparente all'utilizzatore alcune difficoltà nell'invocazione remota
5. Infatti il passaggio di parametri tra sistemi eterogenei richiede
6. un'operazione di marshalling e unmarshalling dei dati, così da consentire di gestire dati rappresentati in maniera diversa, e
7. uno streaming dei dati per appiattire in una stringa eventuali informazioni strutturate
8. I parametri inoltre possono essere passati o
9. per valore, oppure
10. per riferimento, a seconda che ne venga passata
11. Una copia, oppure
12. Un puntatore all'oggetto originario.
13. Un'ulteriore difficoltà da considerare, è che, trattandosi di sistemi distribuiti, sono per loro natura instabili,
14. Un crash tanto di una macchina
15. quanto della rete può propagarsi all'intera struttura

RMI (RMI-IIOP) e CORBA

9

Remote Method Invocation

- Middleware nativo per il mondo JAVA

RMI-IIOP (Internet-Inter-ORB-Protocol)

- Compatibilità con CORBA

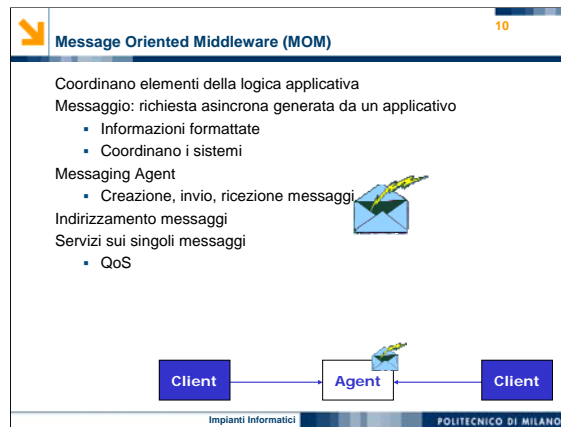
CORBA (Common Object Request Broker Architecture)

- IDL (Interface Definition Language)

Impianti Informatici
POLITECNICO DI MILANO

RMI, RMI-IIOP e CORBA sono dei middleware ad oggetti

1. RMI, acronimo di Remote Method Invocation, è la modalità
2. nativa di Java per la comunicazione tra oggetti distribuiti,
3. RMI-IIOP è un'estensione di RMI che è compatibile con CORBA, un altro middleware ad oggetti, permettendone l'integrazione nel proprio sistema.
4. RMI è infatti strettamente legato al mondo Java, mentre
5. CORBA è un middleware molto più complicato e complesso, non vincolato a particolari piattaforme. È infatti basato
6. su un linguaggio di definizione delle interfacce, detto IDL, che rappresenta gli oggetti distribuiti in un layer di astrazione più di alto rispetto al linguaggio usato nell'implementazione.



- I middleware message-oriented sono diventati un componente essenziale per l'integrazione di operazioni all'interno di un'azienda,
1. consentendo di coordinare i diversi elementi della logica applicativa in modo affidabile e flessibile.
 2. Il termine *messaggio* è usato in diversi ambiti dell'information technology con molteplici significati. Nel nostro specifico caso un messaggio è una richiesta, oppure una risposta o un evento, asincroni, generati da un applicativo.
 3. Essi contengono informazioni utili, precisamente formattate, descriventi la specifica azione,
 4. per coordinare le diverse parti di questi sistemi.
 5. Il sistema message-oriented è di tipo peer-to-peer: in generale, un client può spedire messaggi verso diversi client, così come può riceverne;
 6. per farlo si connette ad un agente che opera da intermediario
 7. offrendo le funzionalità per la creazione, l'invio e la ricezione di messaggi.
 8. Il sistema consente l'indirizzamento dei messaggi, eventualmente con capacità di broadcasting verso diverse destinazioni.
 9. Esso offre inoltre, a seconda dell'implementazione, molteplici servizi e opzioni che possono essere selezionate per specifici messaggi:
 10. un esempio è la qualità del servizio che si vuole ottenere, scegliendo quindi tra il semplice best-effort fino a garanzie di consegna più raffinate.


➤
MOM: Point-To-Point vs Publish-And-Subscribe
11

Point-To-Point (PTP)

- Funzionamento a code
 - Un client invia un messaggio verso una coda specifica
- Simile a mailbox
 - Unica coda per tutti i messaggi

Publish-And-Subscribe (Pub/Sub)


- *Topic*: è un **message broker**
 - I client pubblicano i messaggi
 - I client si sottoscrivono ai messaggi
- Simile a newsgroup



Impianti Informatici
POLITECNICO DI MILANO

Possiamo classificare i middleware message oriented in due tipologie:

1. Quelli di tipo point-to point e
2. quelli publish and subscribe.
3. I primi
4. lavorano come un sistema a code:
5. Quando un client vuole mandare un messaggio ad un altro client, questo viene inviato verso una specifica coda.
6. Il funzionamento è quindi simile a quello di una casella di posta; solitamente
7. si ha un'unica coda per tutti i messaggi in arrivo di qualsiasi natura siano. È cmq possibile una gestione più raffinata, che richiede però implementazioni più complesse.
8. I sistemi publish and subscribe sono invece basati sul concetto di
9. topic, ovvero di un particolare nodo che funge da intermediario per i messaggi;
10. I client pubblicano i messaggi su questo nodo e si
11. sottoscrivono ad esso per ricevere quelli inviati da altri.
12. Ha un funzionamento simile al newsgroup


Java Messages Service (JMS)
12

Message Oriented Middleware

- Interfacce e semantica associata




Sia PTP che Pub/Sub

Applicazione JMS:

- JMS client
- Non-JMS client
- Messaggi
- JMS provider
- Oggetti amministrati

Limitazioni:

- Security
- Load-balancing/fault tolerance
- Triggering client
- Message type repository

Impianti Informatici
 POLITECNICO DI MILANO

1. Java Message Service è l'insieme delle interfacce e della semantica loro associata che definiscono come un client può accedere a servizi message-oriented
2. Consente di realizzare sistemi sia basati sul modello point-to-point, sia su quello publish and subscribe; è anche possibile combinarli assieme, anche se è una pratica poco adottata.
3. Una applicazione JMS è composta da
4. Client JMS, gli applicativi che inviano e ricevono i messaggi
5. Client non JMS, altri applicativi che usano differenti sistemi di messaging
6. I messaggi scambiati tra i client per inviarsi le comunicazioni
7. Il JMS provider, che implementa il middleware e le funzionalità offerte
8. Una serie di oggetti creati e preconfigurati da un amministratore a disposizione dei client.
9. Le principali limitazioni di JMS sono:
10. La mancanza di meccanismi per il controllo di integrità dei messaggi e della privacy.
11. Non c'è supporto per multipli client che accedono a servizi critici in cui è richiesta affidabilità
12. Non sono previsti eventuali meccanismi di triggering, per eventuali client che non sono sempre attivi, ma si innescano a certe condizioni
13. Non esiste uno storage per memorizzare le definizioni dei tipi di messaggio, né un linguaggio che le permetta

Servlet

13

Applicazioni lato server basate su Java

Richiedono un particolare componente un servlet-container (o servlet-engine)

- Appartiene ad un *application server*
- Funzionalità a disposizione dell'applicazione

Ciclo di vita

- Temporanee: vengono istanziate nel momento della richiesta e distrutte al termine della richiesta
- Permanenti: istanziate all'avvio del server e distrutte solo quando viene spento

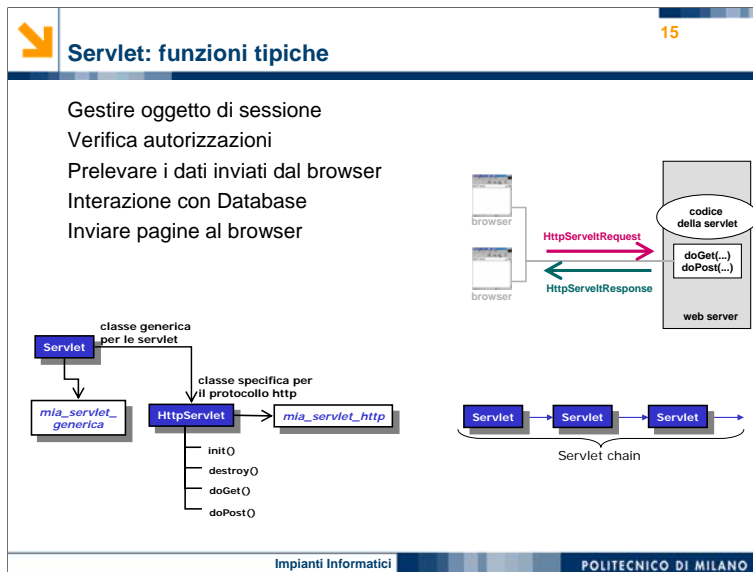
Impianti Informatici
POLITECNICO DI MILANO

Le servlet sono degli

1. applicativi basati su tecnologia Java, generalmente dal codice di limitata dimensione, che vengono eseguiti sul lato server di una connessione Web.
2. Dato che la servlet è un blocco di codice java, un semplice web server non può essere in grado di eseguirla, perché è capace di gestire solo le richieste HTTP. È perciò richiesto un particolare componente, detto container, tramite il quale viene fornito un apposito ambiente virtuale in cui eseguire la servlet.
3. I container, nel caso specifico i servlet container, detti anche servlet engine, sono, in generale, una parte di un particolare server, denominato application server.
4. Essi offrono, oltre alla virtual machine su cui eseguire il codice dell'applicativo, una serie di funzionalità, come, ad esempio,
5. la gestione del ciclo di vita di una servlet, dalla creazione alla distruzione. A seconda del ciclo di vita, possiamo distinguere tra servlet
6. temporanee, se vengono istanziate per servire una singola richiesta, oppure
7. permanenti se il loro ciclo di vita dura per tutto il periodo di attività dell'application server

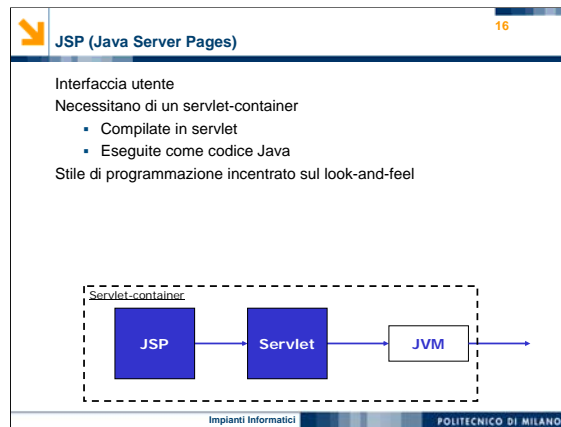
Servlet vs CGI	
Servlet <ul style="list-style-type: none"> JVM sempre attiva Un thread per ogni richiesta Singola copia del codice Servlet in esecuzione 	CGI <ul style="list-style-type: none"> Un processo per ogni richiesta Multiple istanze Difficile operazioni su persistenza dati
Migliori prestazioni Platform-independent Librerie Java a disposizione <ul style="list-style-type: none"> Applet Database Rmi ... 	

1. Le servlet hanno sostituito le
2. CGI, le Common Gateway Interface
3. In esse, per ogni richiesta HTTP veniva fatto partire un nuovo processo, così, se ad es., il codice da eseguire era particolarmente corto, l'overhead x inizializzare il processo poteva dominare il tempo di esecuzione.
4. Con le servlet, la JVM è sempre attiva e pronta a gestire le richieste utilizzando
5. un thread Java relativamente leggero in termini computazionali.
6. Un altro vantaggio sta nel fatto che con CGI, all'arrivo di N richieste simultanee allo stesso programma CGI, venivano caricate in memoria N istanze della stessa applicazione;
7. con le servlet ci sarebbero N thread, ma
8. una singola copia del codice della servlet.
9. Un'altra problematica delle applicazioni CGI era il fatto che era difficile l'esecuzione di operazioni inerenti la persistenza dei dati, quali mantenere aperta una connessione al DB. Il problema è dovuto al fatto che quando il programma ha risolto la richiesta viene chiuso.
10. Le servlet rimangono invece in memoria, così rendono molto più agevoli operazioni come ad esempio il salvataggio di dati sul db una volta esaudita la richiesta.
I vantaggi delle servlet, rispetto alle CGI, sono quindi in termini di
11. Migliori prestazioni,
12. Di essere platform-independent perché basate su java, e
13. di avere a disposizione le molteplici librerie java, potendo, ad esempio, comunicare con un'applet, interrogare un database, accedere ad un oggetto remoto tramite rmi.



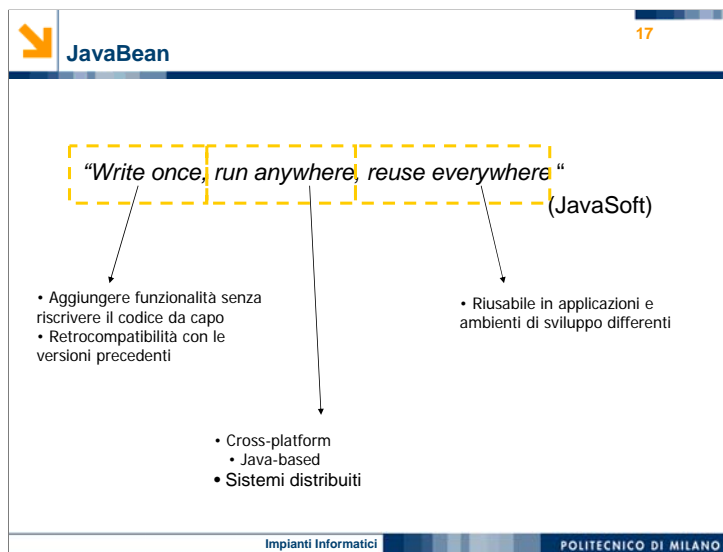
Le servlet vengono utilizzate all'interno delle web application con compiti differenti.

1. Generalmente si creano catene di servlet, in cui ciascun elemento esegue un preciso compito
2. Della generica classe java servlet, esiste una particolare sottoclasse
3. detta HttpServlet, specifica per il protocollo http,
4. con metodi appositi per gestire le richieste http. Il programmatore può quindi implementare
5. sia generiche servlet, sia
6. servlet che intercettano le richieste http. Le funzioni di ciascuna servlet possono essere:
7. Prelevare le informazioni di stato dall'oggetto di sessione
8. Verificare le autorizzazioni prima di eseguire la servlet
9. Raccogliere i dati inviati dal browser al server, ad esempio, mediante una form
10. Eseguire una query/update su un database
11. Inviare al browser una pagina con il risultato dell'operazione sul database



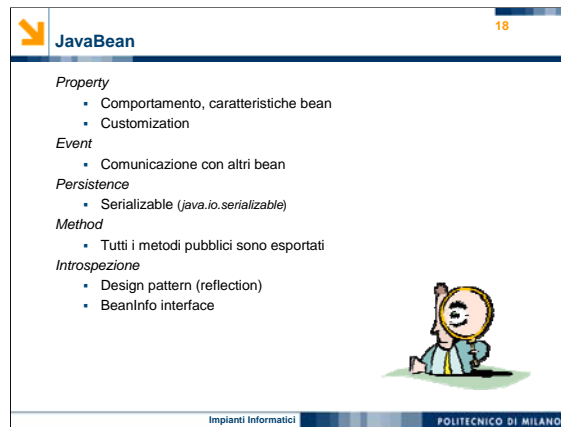
Parlando di servlet non si possono non menzionare le pagine JSP, uno dei componenti principali

1. Usato per la visualizzazione dell'interfaccia utente.
2. Le JSP vengono compilate in Servlet dal container, senza il quale non possono essere utilizzate,
3. e poi eseguite come codice java. La differenza è che non sono scritte in codice java puro,
4. ma lo stile di programmazione è incentrato sull'interfaccia, piuttosto che sulla business logic.



I javabeen sono una pubblica API pubblicata da SUN. Sono delle classi java, realizzate seguendo uno specifico standard.

1. I java bean sono stati pensati per venire in contro alle esigenze dei programmatori, semplificandone lo sviluppo di applicazioni.
2. Write once si riferisce al bisogno del codice javabeen di essere scritto una volta sola, e non richiedere riscritture per
3. aggiungere o migliorare le funzionalità. Spesso i programmatori sono infatti costretti a riscrivere il codice quando decidono di apportare dei cambiamenti, con costose perdite di tempo e il rischio di manomettere parti funzionanti.
4. Un altro punto considerato è il controllo delle versioni, incoraggiando così gli sviluppatori a fare cambiamenti alle proprie componenti software in maniera incrementale; il risultato è un incremento di funzionalità, con retrocompatibilità con le precedenti versioni.
5. Run anywhere si riferisce alla capacità dei componenti Javabeen
6. di essere eseguiti in un qualsiasi ambiente. Una tecnologia di questo tipo ha infatti maggiori chance di successo, visto che il target di utilizzatori potenziali è sempre più eterogeneo.
7. I javabeen lo sono per loro natura dato che sono basati su java. Run anywhere indica anche
8. la possibilità di esecuzione in ambienti formati da sistemi distribuiti.
9. Reuse Everywhere suggerisce la capacità dei Javabeen di essere
10. riutilizzati in diversi scenari, quali differenti applicazioni, applet, oppure siti web



Le caratteristiche di un javabeen, e che lo distinguono da una comune classe java, seguono un preciso standard:

1. Le property determinano
2. le caratteristiche del bean, che possono essere modificate in corso di design;
3. affinché ciò sia possibile, i bean espongono pubblicamente le loro property
4. I bean usano gli eventi
5. per comunicare con altri bean. Un bean che vuole ricevere determinati eventi, detto listener bean, si registra sul bean sorgente, detto source bean.
6. I javabeen devono inoltre essere oggetti che sia possibile rendere persistenti, cioè di cui si possa salvare e ripristinare lo stato corrente, mantenendo intatte le property.
7. La persistenza è supportata dalla serializzazione; i java bean implementano infatti l'interfaccia serializable
8. I metodi dei bean non sono differenti dai comuni metodi java.
9. Per default tutti i metodi pubblici vengono esportati.
10. L'introspezione è il processo tramite il quale vengono scoperte le caratteristiche del bean. Esistono due modalità per supportare l'introspezione:
11. una sfrutta l'adesione dei bean ad un design pattern, cioè ad un insieme di regole da adottare per i nomi delle proprietà, dei metodi e degli eventi. Tramite un meccanismo simile alla reflection si ottengono le caratteristiche del bean.
12. L'altro metodo prevede la creazione di un'apposita classe informativa che implementi l'interfaccia BeanInfo

➤
EJB (Enterprise Java Bean)
19

Un *Enterprise Bean* è un componente software server-side. Può essere composto di uno o più oggetti, a fronte di un'unica interfaccia con cui il *client* può interagire.

- Catena di EJB

Necessitano di un apposito *container*, e quindi di un *application server EJB-compliant*

- Ambiente per EJB
- Persistenza, transazioni, sicurezza, connessioni
- Accesso alle risorse esterne (DB,..)

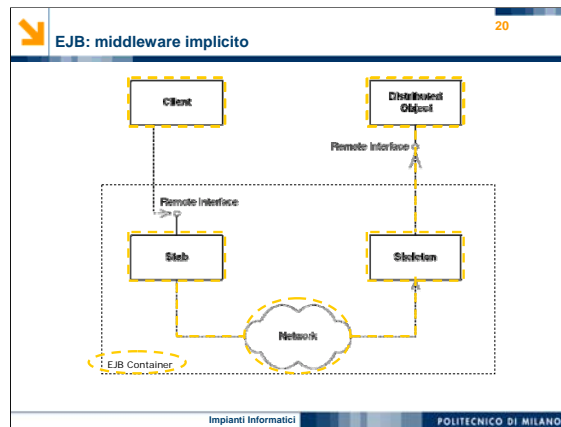
```

graph LR
    EJB1[EJB] --> EJB2[EJB]
    EJB2 --> EJB3[EJB]
    EJB3 --> Exit[...]
    subgraph EJB_chain [EJB chain]
        EJB1
        EJB2
        EJB3
    end

```


Impianti Informatici
POLITECNICO DI MILANO

1. Gli EJB sono componenti software lato server
2. Il client di un EJB potrebbe essere qualsiasi cosa, da una servlet, ad un'applet o anche un altro *enterprise bean*. In quest'ultimo caso, una richiesta ad un bean può risultare in
3. una catena di bean da chiamare; questo è un meccanismo molto potente perché permette di suddividere compiti complessi, permettendo ad un bean di chiamare una varietà di bean pronti a gestire un particolare task.
4. Così come le servlet, necessitano di un apposito container. Il container, ha il principale compito di fornire un
5. ambiente in cui l'enterprise bean possa girare. Esso lo rende disponibile ai client x essere invocati remotamente. In sostanza il container si interpone tra client e EJB, gestendo la connessione tra client e bean,
6. gestendo transazioni, fornendo persistenza, sicurezza, pool di connessioni
7. L'accesso a risorse esterne, quali DB, legacy system, è gestito dal container; ciò significa che il programmatore non deve preoccuparsi di allocazione e deallocazione di tali risorse, ma è il container che le gestisce e ne garantisce l'accesso



Gli EJB sono a tutti gli effetti un middleware implicito. Il *middleware* fornisce ad un'applicazione distribuita appositi servizi di integrazione con la rete di comunicazione.


1. Un oggetto distribuito è un oggetto che può essere chiamato da un sistema remoto.
2. Il client chiama uno *stub*,
3. responsabile di mascherare la rete di comunicazione; lo stub è a conoscenza di
4. come operare sulla rete usando le socket
5. Lo stub chiama uno *skeleton*, lato server, che maschera la rete dall'oggetto remoto distribuito. Lo skeleton è in grado di ricevere una chiamata su una socket e traslare i dati provenienti dalla rete in una rappresentazione adatta a Java.
6. Lo skeleton chiama l'oggetto distribuito,
7. il quale esegue il proprio lavoro e ritorna il controllo allo skeleton, il quale risponde allo stub e da questi il controllo ritorna al client.
8. Stub e oggetto distribuito implementano la stessa interfaccia remota, questo significa che il client vede l'oggetto distribuito come se fosse in locale (*trasparenza*)
Gli EJB sono un middleware **implicito**, nel senso che non bisogna scrivere API del middleware, ma è sufficiente creare l'oggetto distribuito, il quale contiene la *business logic*;
9. quindi si dichiarano i servizi che l'oggetto deve avere, quali la persistenza..
10. e mediante un apposito tool fornito si genera un oggetto detto *request interceptor*, usato x intercettare le richieste del client.

 **EJB: Session Bean** 21

Modellano un processo, un'azione

- Accedere ad un DB
- Collegarsi ad un legacy system
- Chiamare altri EJB.

Solitamente non persistente



Impianti Informatici **POLITECNICO DI MILANO**

Esistono 3 tipi di EJB: i session, gli entity e i message-driven bean

1. I **Session Bean** modellano un processo, un'azione, un algoritmo,
2. quindi parti della business logic. L'azione può essere qualsiasi cosa,
3. come accedere ad un DB,
4. collegarsi ad un legacy system,
5. chiamare altri EJB.

Ad esempio possono eseguire il calcolo del prezzo di un prodotto, la gestione di un workflow, quindi di un insieme di attività, oppure gestire un catalogo...


6. Sono solitamente transienti, cioè non persistenti, con un ciclo di vita breve, e non sopravvivono ad eventuali crash del server

➤
EJB: Entity Bean
22

Modellano i dati
 Fungono da contenitore per le informazioni di un database

- Un prodotto
- Un ordine
- Un operaio
- Una carta di credito

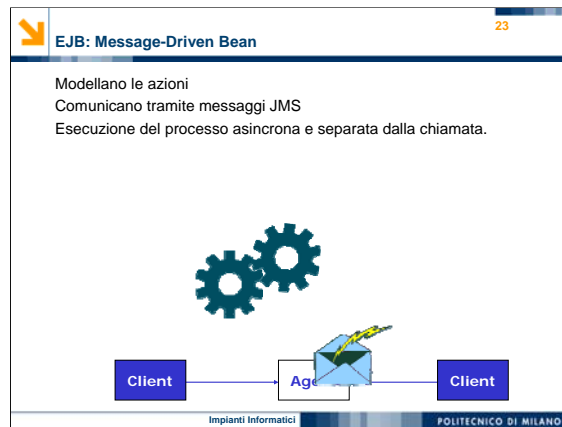
Sono usati dai *session bean*
 Gestiscono la persistenza



Impianti Informatici
POLITECNICO DI MILANO


Gli Entity Bean

1. modellano i dati dell'applicazione.
2. In pratica sono oggetti che fungono da contenitore per le informazioni contenute in un DataBase. Un entity bean corrisponde grossomodo ad una entry di un database; può essere, ad esempio:
3. 1 prodotto,
4. 1 ordine,
5. 1 operaio,
6. 1 una carta di credito...
7. I session bean si avvalgono solitamente degli Entity bean x svolgere il loro compito.
8. Sono solitamente persistenti, con un lungo ciclo di vita, così da sopravvivere ad eventuali crash del server. La caratteristica principale è la loro capacità di rendersi persistenti in modo autonomo memorizzandosi in uno storage permanente, come un database o un sistema legacy



I **Message-Driven Bean** sono

- simili ai Session Bean perché anch'essi modellano le azioni.
- La differenza tra i due è che è possibile chiamare un Message-Driven Bean inviandogli un messaggio, come avviene con JMS,
- e quindi l'esecuzione del processo può essere asincrona e separata dalla chiamata.


Message-Driven Bean: caratteristiche
24

Non hanno interfaccia locale o remota

- Comunicano solo con messaggi
- Qualsiasi client che implementi JMS può usarli

onMessage()

- Analisi a run-time del messaggio


Nessun valore di ritorno

Nessuna *eccezione* verso il client

- Il container cattura *eccezioni* di sistema

Stateless

Conservazione dei messaggi per destinatari non pronti



Impianti Informatici
POLITECNICO DI MILANO

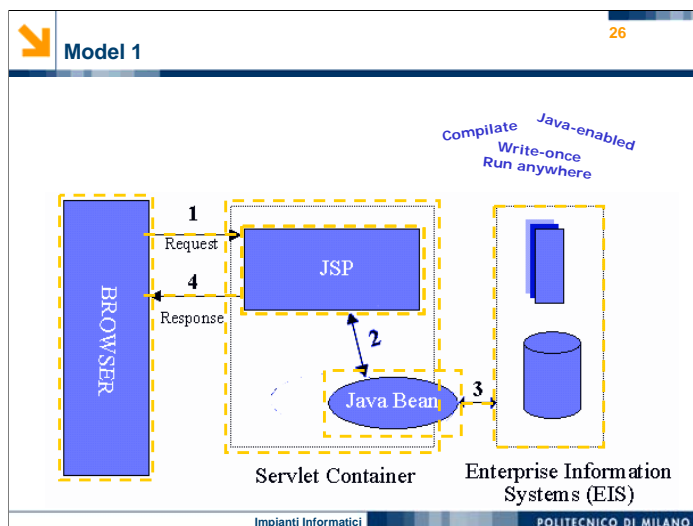
1. I message-driven, a differenza degli altri bean, non hanno né un'interfaccia locale, né una remota.
2. Essi vengono chiamati tramite messaggi, non invocando un metodo.
3. In questo modo sono compatibili con qualsiasi client che usi l'interfaccia JMS
4. Implementano, solitamente, un unico metodo, *onMessage()*, che accetta un messaggio JMS.
5. Occorre controllare a runtime il tipo di messaggio e l'azione da fare
6. Non hanno alcun valore di ritorno, perché sono disaccoppiati da chi ha inviato il messaggio; la risposta sarà un altro messaggio.
7. Analogamente non può sollevare eccezioni verso il client;
8. esistono eccezioni di sistema catturate dal container
9. Sono stateless, quindi ogni messaggio è indipendente dagli altri
10. Permette di conservare i messaggi in attesa che il destinatario sia attivo

Impianti Informatici

 POLITECNICO DI MILANO

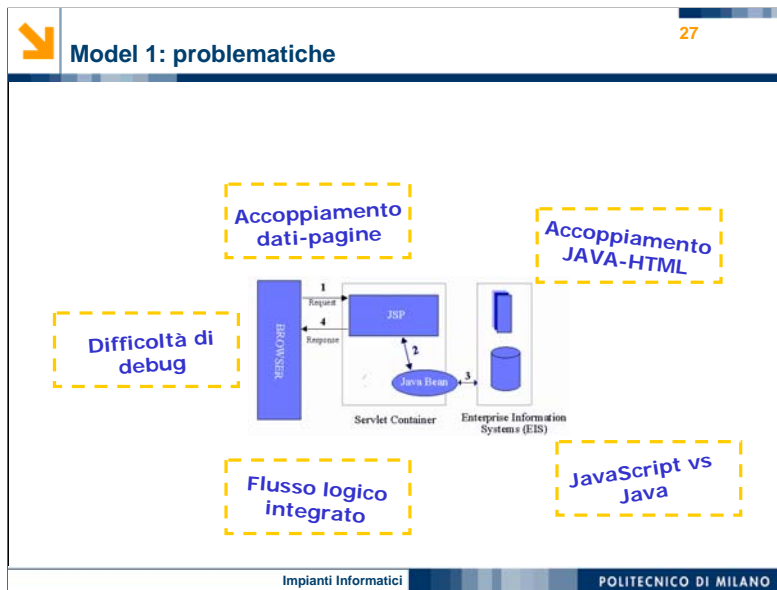


Web application - tecnologie



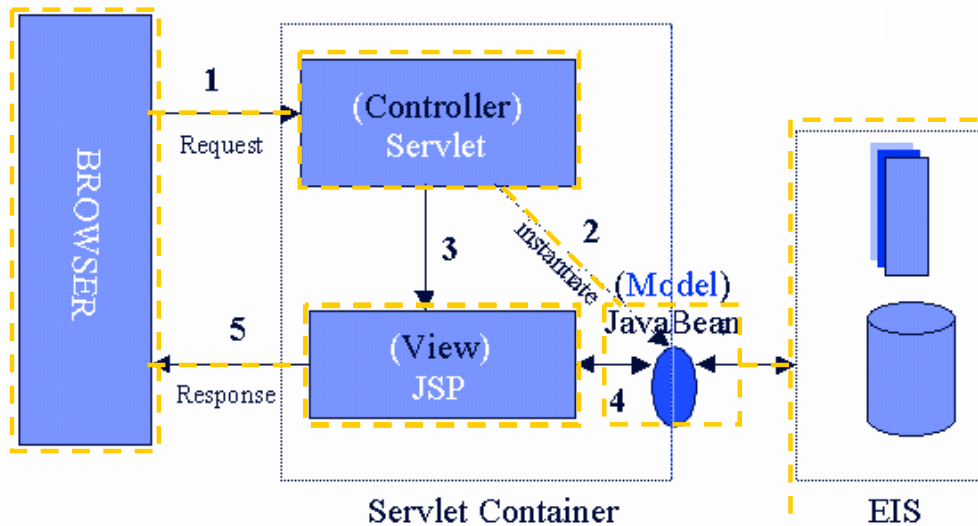
La realizzazione di una Web Application, sia che essa avvenga mediante tecnologia J2EE, sia con tecnologia .NET, può seguire differenti modelli di programmazione, a seconda di come avviene la divisione logica dei componenti. L'analisi di tali modelli avverrà principalmente riferendosi ad una delle due tecnologie, quella J2EE, tenendo comunque conto dell'esistenza di componenti duali in quella .NET.

1. Quando sono state introdotte le JSP, il principale paradigma
2. di riferimento era il Model 1. L'uso delle jsp, rispetto ad altre tecnologie, presenta i vantaggi che le pagine:
3. sono compilate, e non interpretate, quindi con prestazioni migliori
4. Consentono di accedere in modo completo al linguaggio java
5. Condividono con java le caratteristiche Write Once, Run Anywhere.
6. Le JSP, dato che vengono tradotte in servlet, necessitano di un servlet-container per essere eseguite.
7. L'architettura del framework ingloba nelle pagine JSP il processamento delle richieste in ingresso,
8. e la conseguente risposta al client per proporre l'output all'utente. È la pagina jsp stessa che seleziona la pagina successiva da visualizzare, andando ad analizzare la request dell'utente e gli eventuali parametri ricevuti.
9. Vi è una separazione dalla presentazione al contenuto, perché tutti i dati necessari prevedono l'uso di javabean, i quali
10. contengono i dati prelevati dal sistema informativo aziendale.



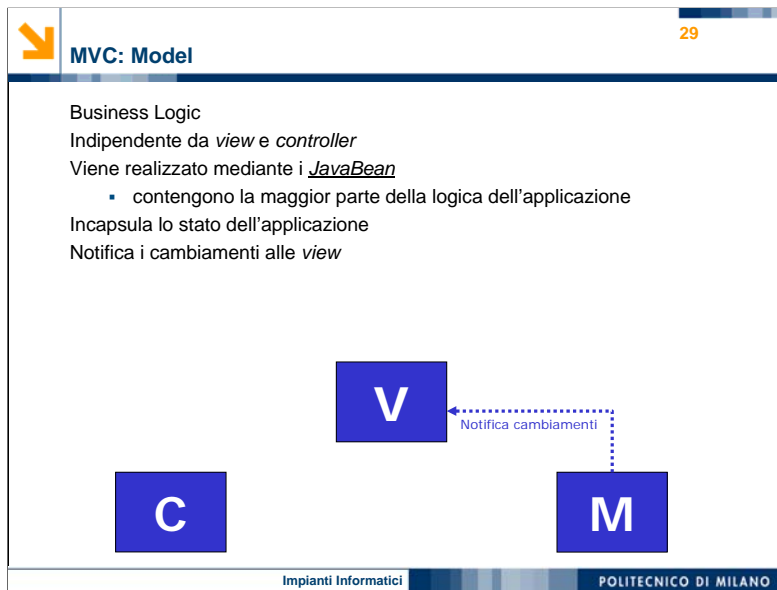
Le principali limitazioni del Model 1, sono

1. il forte accoppiamento tra codice Java e codice HTML, lo sviluppatore dell'applicazione deve essere tanto un buon programmatore, quanto un buon web designer
2. C'è tendenza a mischiare codice Javascript, ad esempio per la validazione dei campi, con conseguente divisione poco chiara tra la parte java e la parte javascript
3. Per capire l'intero flusso dell'applicazione bisogna "navigare" tutte le pagine che la compongono
4. E' difficile procedere nella fase di debugging del codice quando si hanno 3 linguaggi in un unico contesto, cioè HTML, Java e JavaScript
5. Dati e pagine sono strettamente accoppiati, quindi il cambiamento delle strutture dati dell'applicazione spesso provoca una revisione di ogni pagina che la compone

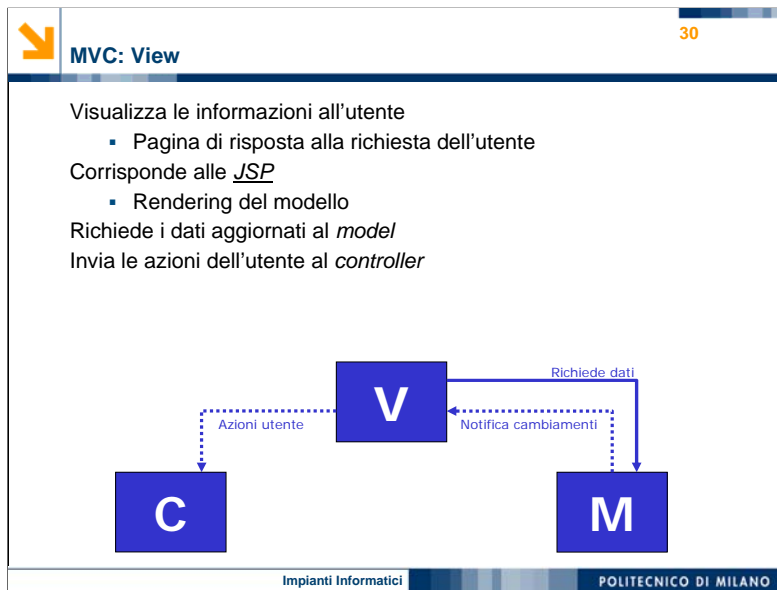


L'evoluzione del Model 1 è

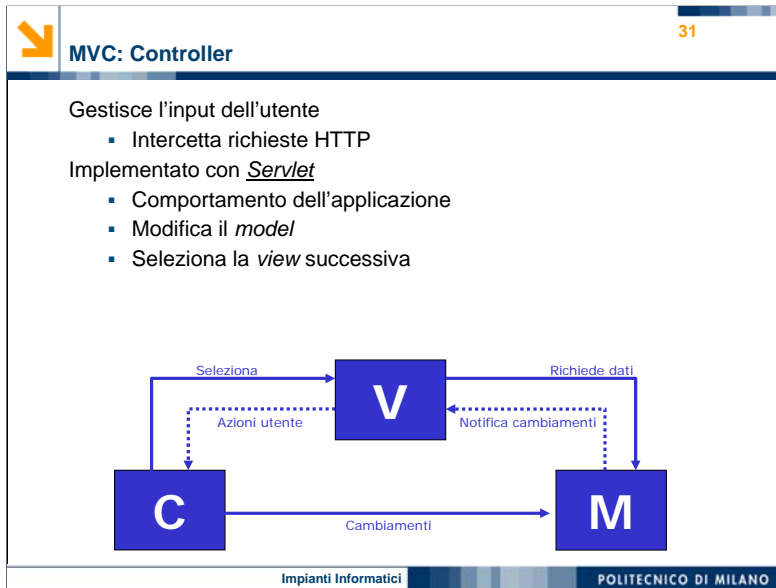
1. il Model 2, detto anche Model View Controller. In sostanza si avvantaggia dei punti di forza predominanti di due tecnologie,
2. usando le JSP per generare il presentation layer e
3. le servlet per compiti di processamento.
4. La richiesta del client è prima intercettata dalla *servlet*, denominata **controller servlet**; essa si occupa del processamento della richiesta.
5. Il controller è inoltre responsabile della creazione di javabeen o di altri oggetti usati dalle jsp, contenente
6. i dati prelevati dal sistema informativo aziendale, e determina quale pagina andare a visualizzare successivamente.
7. La richiesta del client perciò non viene mai inviata direttamente ad una pagina JSP. Questo permette alla servlet di effettuare operazioni quali *autenticazione, autorizzazione, logging, internazionalizzazione dei contenuti...*
8. La pagina JSP viene quindi inviata al client dove viene visualizzata



1. Il Modello corrisponde
2. alla logica e ai dati dell'applicazione
3. Il codice e i dati corrispondenti sono concettualmente indipendenti dalla view e dal controller
4. Viene realizzato tipicamente mediante l'uso dei Javabean,
5. che espongono le funzioni principali dell'applicativo.
6. Il model incapsula inoltre lo stato dell'applicazione
7. ed è in grado di notificarne i cambiamenti alle view



1. La View è la parte dell'applicazione che si
2. occupa di visualizzare le informazioni all'utente.
3. In una web application è la visualizzazione della pagina di risposta alle richieste dell'utente.
4. Corrisponde alle JSP ed effettua
5. il rendering del modello,
6. richiedendone i dati aggiornati.
7. Invia inoltre le azioni dell'utente al controller, permettendogli, in base ai parametri della request, di selezionare la view successiva.



1. Il Controller è la parte dell'applicazione
2. responsabile di gestire l'input dell'utente
3. In un'applicazione web l'input corrisponde ad una richiesta HTTP.
4. Nella tecnologia J2EE è rappresentato dalle *Servlet*.
5. Il controller definisce il comportamento dell'applicazione,
6. mappando le azioni utente in cambiamenti nel modello.
7. Seleziona quindi la view successiva per la risposta al client

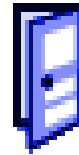


Facilità sviluppo e gestione applicazioni

- Netta separazione tra *business logic*, *presentation* e *request processing*

Unico punto di ingresso: il *controller*

- Le applicazioni sono generalmente più semplici da mantenere e più facilmente estendibili
- Sicurezza, validazione dell'input, internazionalizzazione



I vantaggi del modello MVC riguardano la maggiore

1. facilità di sviluppo e mantenimento delle applicazioni, così come l'estendibilità delle stesse, dato che i diversi
2. componenti hanno chiare e distinte responsabilità all'interno dell'architettura. La differenza principale rispetto all'approccio model 1, è che l'introduzione della servlet che funge
3. da controller è l'**unico punto di ingresso**, dove viene accentrato tutto il controllo,
4. andando ad incoraggiare il riuso e l'estensibilità dell'applicazione.
5. Eventuali operazioni che impattano su tutta l'applicazione, come l'aggiunta di sicurezza, la validazione dell'input o l'internazionalizzazione dei contenuti, possono essere inserite all'interno del controller, quindi in un unico punto, velocizzandone e semplificandone l'integrazione.

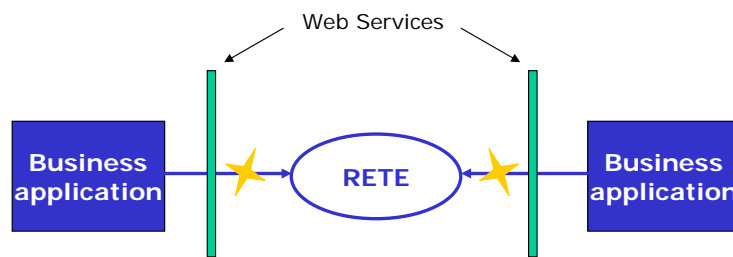


Servizi offerti via Web

- Applicazioni B2B

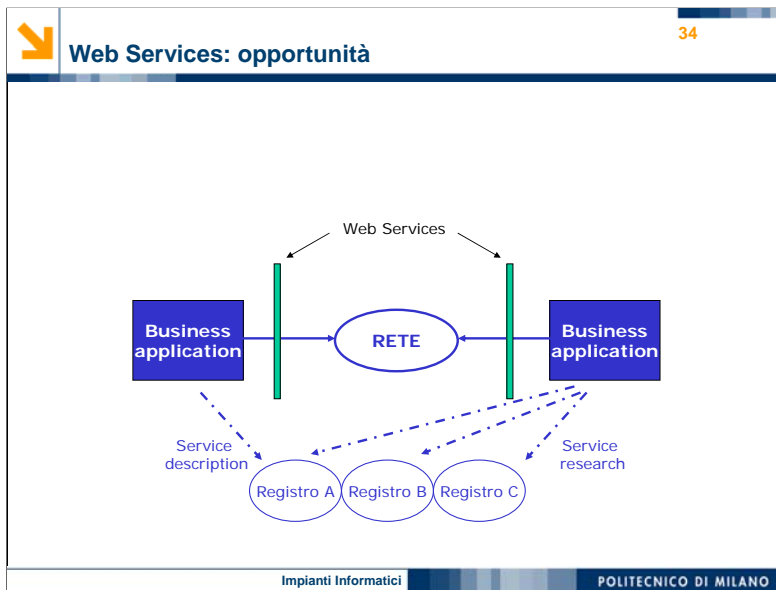
Sono una nuova tecnologia di *middleware distribuito* basata su XML (Extensible Markup Language)

- Opportunità di integrazione tra le applicazioni business



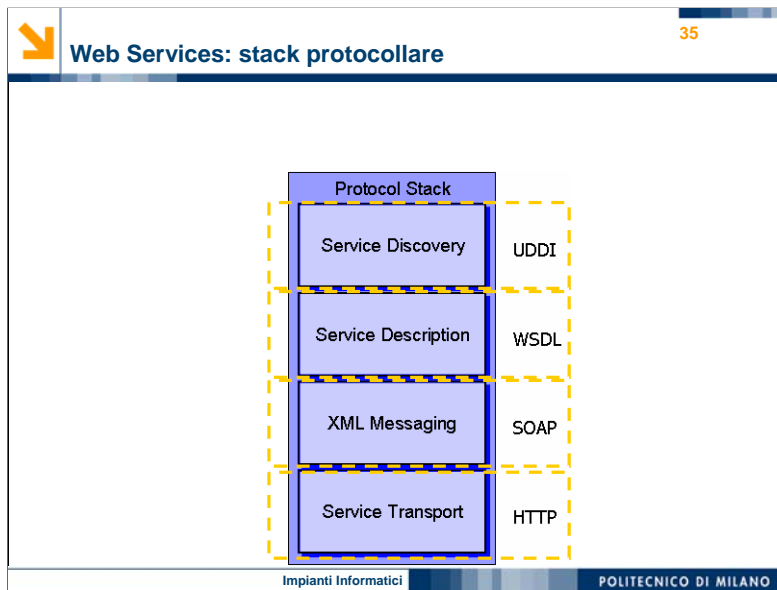
I Web Services, nel significato più generale del termine,

1. sono dei servizi offerti via web, generalmente rivolti alle
2. transazioni business-to-business.
L'avvento dei web services non è tanto una rivoluzione nelle tecnologie utilizzate,
3. ma la naturale evoluzione delle applicazioni basate su XML. La vera rivoluzione è nelle opportunità offerte dai web services.
4. Prima del loro avvento, l'integrazione tra le applicazioni aziendali era resa difficile dai
5. diversi linguaggi di programmazione e middleware usati all'interno delle singole organizzazioni.
6. Con i web services, ogni applicazione può essere integrata alle altre tramite Internet.



Il contesto che si può immaginare con l'uso dei web services, è quello in cui

1. i servizi, ovvero le applicazioni, sono registrati in
2. appositi registri, pubblici o privati. Questi servizi sono autoesplicativi in termini di
3. funzionalità offerte, di interfaccia, di requisiti richiesti, di modalità e di condizioni per l'utilizzo.
4. I potenziali utilizzatori di questi servizi, possono effettuare
5. ricerche a diverso dettaglio e disporre delle descrizioni specifiche di ogni web service, per verificare se le caratteristiche proposte sono attinenti alle proprie necessità.



Lo stack protocollare dei web services si basa su tecnologie preesistenti e ben standardizzate.

- Il livello più alto riguarda il *Service Discover*, un sistema di naming e discovery dei web services.
- Il livello *Service Description* implementa un apposito linguaggio per la definizione dell'interfaccia dei servizi web.
- Il livello di *Messaging* utilizza *SOAP* come protocollo di messaggistica basato su XML.
- Il punto di forza dei web services è l'integrazione con l'HTTP, il protocollo di comunicazione utilizzato su Internet

➤

SOAP (Simple Object Access Protocol)

36

Lightweight protocol

- Modalità di passaggio di dati in XML

Interfaccia con HTTP

Indipendenti da


- sistema operativo
- protocollo di comunicazione
- linguaggio di programmazione

```

graph TD
    subgraph Protocol_Stack [Protocol Stack]
        SD[Service Discovery]
        SD --- SDL[Service Description]
        SDL --- XM[XML Messaging]
        XM --- ST[Service Transport]
    end
    SD --- UDDI[UDDI]
    SDL --- WSDL[WSDL]
    XM --- SOAP[SOAP]
    ST --- HTTP[HTTP]
    style XM stroke-dasharray: 5 5
    
```

Impianti Informatici
POLITECNICO DI MILANO

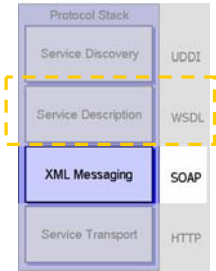
1. Il SOAP è un protocollo leggero che definisce la
2. modalità di passaggio di dati codificati in XML
3. Funge da interfaccia con il layer di comunicazione sottostante, tipicamente HTTP.
4. I messaggi soap sono indipendenti dal sistema operativo,
5. dal protocollo di comunicazione e
6. dal linguaggio di programmazione utilizzati.

**WSDL (Web Services Definition Language)**37

Interfaccia servizi

Modalità e protocolli di accesso

- Parametri in ingresso



Protocol Stack	
Service Discovery	UDDI
Service Description	WSDL
XML Messaging	SOAP
Service Transport	HTTP

Impianti InformaticiPOLITECNICO DI MILANO

WSDL è un linguaggio che permette la

- specifica dell'interfaccia dei servizi disponibili
- Indica le modalità ed i protocolli di accesso a tali servizi,
- così come i parametri in ingresso richiesti dall'applicativo

➤
UDDI (Universal Description Discovery and Integration)
38

Sistema di directory distribuito

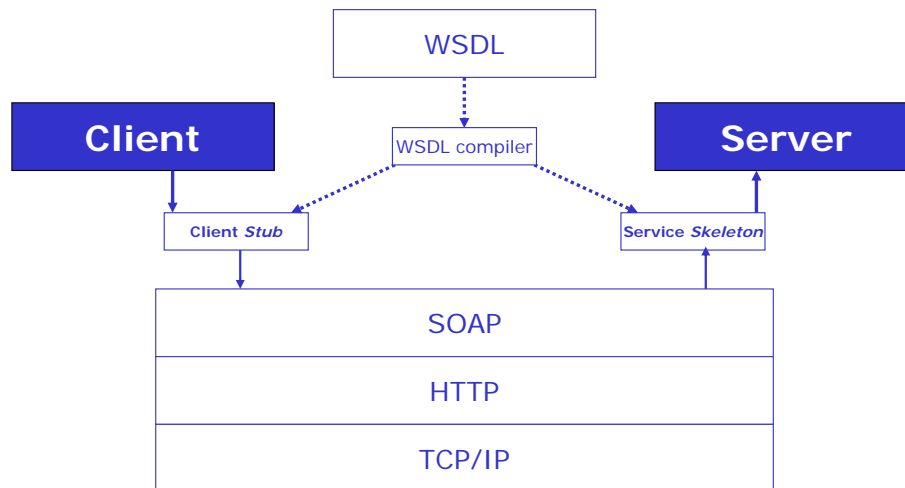
- Registri pubblici
- Registri privati

Tipi di registri

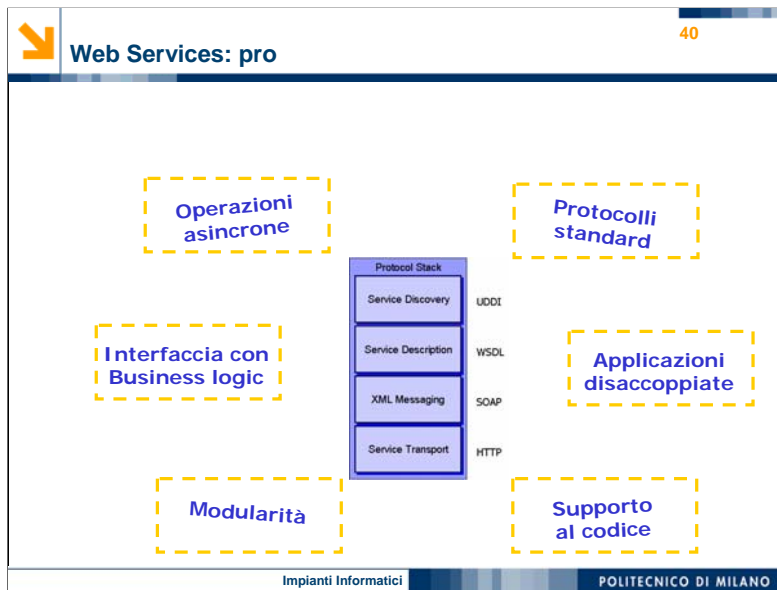
- White pages
 - Informazioni sul provider
- Yellow pages
 - Tassonomia standard dei servizi e delle organizzazioni registrate
- Green pages
 - Informazioni tecniche sui servizi

Impianti Informatici
POLITECNICO DI MILANO

1. UDDI è un sistema di directory distribuito.
2. Ci sono registri pubblici, in cui chiunque può registrare e richiedere un servizio, senza particolari processi di validazioni e
3. registri privati con criteri più restrittivi
 - Esistono 3 tipologie di registri
 - Le White pages, contenenti informazioni di base sul provider del servizio, come nome, descrizione, indirizzo, contatti
 - Le Yellow pages, una classificazione dei servizi e delle compagnie registrate mediante l'uso di tassonomie standard
 - Le Green pages, infine, rappresentano le informazioni tecniche sui servizi, quindi come invocarli, come effettuarne il bind, che tipo di parametri accettano

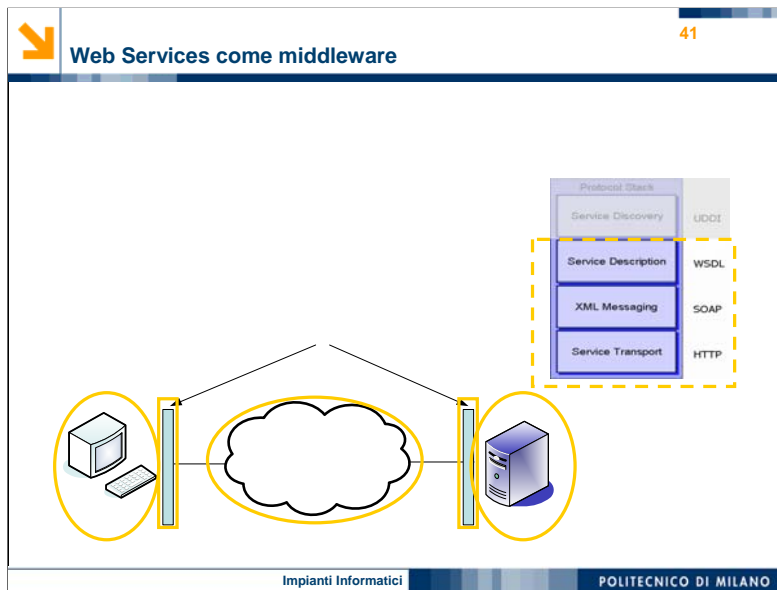


1. La realizzazione di un web service, inizia dalla codifica dell'interfaccia del servizio, mediante il linguaggio WSDL.
2. Mediante un apposito compilatore, si creano
3. lo stub, e
4. lo skeleton.
5. Quando il client vuole usare un
6. servizio offerto da un server
7. Chiama lo stub, che si occupa di impacchettare i parametri del servizio in uno o più messaggi, facendo marshalling e serializzazione dei dati.
8. Il SOAP crea i messaggi XML, che vengono
9. Inviati tramite la rete
10. allo skeleton, che effettua unmarshalling e deserializzazione dei dati e creare la chiamata per
11. invocare il server



Gli aspetti positivi dell'uso dei web services sono:

1. l'utilizzo di protocolli standard, come HTTP, XML, SOAP, UDDI, WSDL, con facile integrazione in piattaforme e sistemi già in uso.
2. Le applicazioni sono poco accoppiate tra loro, con maggiore semplicità di gestione.
3. Nello sviluppo dei sistemi viene facilitata la comunicazione tra gli applicativi, anche differenti tra loro, e il riuso di codice già esistente.
4. Ogni servizio è una componente software indipendente.
5. I web service rappresentano un'interfaccia con la business logic, permettendo l'accesso ad un preciso set di funzionalità
6. Consentono inoltre l'esecuzione di operazioni asincrone.



Nella pratica, l'uso dei web services non è tanto diffuso per la sue promettenti opportunità di realizzare una sorta di mercato dei servizi, in cui i provider si registrano presso registri pubblici, e gli utenti ricercano il web service che svolge la funzione che necessitano.

1. Piuttosto dello stack protocollare vengono usati praticamente i primi due livelli, con scarso utilizzo del sistema UDDI, e l'uso che ne viene fatto è in termini di middleware Web Services
2. Client e server possono perciò usare i web services
3. per comunicare attraverso la rete.
L'estremo vantaggio rispetto ad altri middleware, come RMI o Corba,
4. è che le interfacce dei Web Services comunicano tra loro mediante il protocollo HTTP, ovvero la porta 80 del livello TCP, eliminando la maggior parte dei problemi dovuti alla presenza di apparati, tra client e server, quali router e soprattutto firewall. Rete

Client