Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Tecnologie Web

1) INTRODUZIONE

1.1 Introduzione al web

Il **World Wide Web** (WWW) nasce nel 1994. Il padre del WWW è considerato **Tim Berners-Lee** che tra il 1989 e il 1991 lavorava al CERN. L'idea su cui si basava il WWW è quello di **ipertesto.**

Le componenti chiave di questa prima versione del web sono:

1. La versione iniziale di **HTML** (HyperText Mark-up Language);
2. La versione iniziale di **HTTP** (HyperText Transfer Protocol);
3. **Server web**;
4. **Browser**.

L'applicazione browser è essenzialmente un **client HTTP**, che prepara le relative richieste al server web considerato per poi interpretare le relative risposte.

La risposta HTTP conteneva un documento HTML da visualizzare.

Tra le funzionalità essenziali di un browser sta la visualizzazione di **oggetti web**, che possono essere documenti HTML, ma non solo.

1.1.1 Ipertesti

Normalmente un documento testuale o testo ha una struttura sequenziale, nel senso che si compone di una stringa di caratteri che viene letta dall'inizio alla fine.

Un **ipertesto** è un testo a cui sono stati aggiunti degli archi con direzione o **link** che rappresentano associazioni. Non si presta quindi ad una semplice lettura sequenziale, ma ad essere navigato, nel senso che in diversi punti del testo si presenta la scelta se proseguire la lettura o seguire il link. La direzione indica quindi il verso di lettura, **il link può venir seguito in quel verso ma non nel verso opposto**.

I link possono essere sia **interni**, se sono diretti verso una parte del documento, che **esterni**, se sono diretti ad un diverso documento.

La struttura associata ad un ipertesto è quindi quella di un grafo con direzione, in cui i **nodi** corrispondono a **documenti testuali** e gli **archi** a **link**.

I nodi non si riferiscono necessariamente solo a documenti testuali, ma più in generale a oggetti web, che possono essere immagini, suoni, filmati etc…

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

L'introduzione del concetto di ipertesto ha portato all'**SGML** (Standard Generalized Markup Language), un linguaggio che permette di definire una grammatica per testi, in particolare per il tipo di etichettatura che contengono.

L'HTML è un'istanza dell'SGML.

Possiamo quindi considerare i seguenti quattro linguaggi utili per rappresentare gli ipertesti:

1. **SGML**;
2. **HTML**: HyperTest Mark-up Language;
3. **XML**: eXtensible Markup Language;
4. **XHTML**: eXtensibile HyperTest Markup Language.

L'HTML è un linguaggio di etichettatura per gli ipertesti.

Un browser deve essere in grado di seguire i link. Seguire un link significa preparare una richiesta HTTP, da spedire al server web identificato dal link per ottenere la risorsa desiderata.

I primi browser non avevano un'interfaccia grafica, ma con modalità di linea. Ne esistono ancora: lynx, links, …

1.1.2 Il Consorzio del W3C

Nell'ottobre del 1994, Tim Berners-Lee ha fondato il **W**orld **W**ide **W**eb **C**onsortium (W3C) in collaborazione con il **CERN** e con il supporto del **DARPA** e della **Commissione Europea**. Non si tratta di un produttore di strumenti, ma di protocolli e linee guida atte ad assicurare lo sviluppo a lungo termine del web. Lo sviluppo di nuovi **standard** vede la partecipazione non solo delle organizzazioni membri e dello staff permanente del consorzio, ma anche dell'opinione pubblica attraverso processi di condivisione e sviluppo.

1.1.3 I client HTTP: i browser

Nel **1995** si diffondono i browser Mosaic e Netscape. L'evento importante del **1996** è l'entrata della Microsoft nella lotta dei browser: qualche anno dopo risultò vincitrice con Internet Explorer.

Si apre così la strada ad una moltitudine di nuove applicazioni multimediali, tra cui la trasmissione di video ad alta qualità on-demand e la videoconferenza interattiva ad alta qualità.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

1.1.4 Cosa si intende per World Wide Web?

Il **W**orld **W**ide **W**eb è un'applicazione **client/server** operante su Internet e reti intranet **TCP/IP** (Transmssion Control Protocol/Internet Protocol) attraverso il protocollo **HTTP** (HyperText Transfer Protocol). Le applicazioni web vedono la rete come fornitore di servizi.

1.2 Servizi forniti da Internet

1.2.1 Terminali, client e server

I computer collegati a Internet vengono chiamati in modo equivalente host o terminal o end system. Un'applicazione **client** è un'applicazione che usa un servizio da un programma **server**. Client e server possono venir eseguiti su macchine diverse oppure sulla stessa macchina. Di solito, vengono eseguite su macchine differenti: sono quindi **applicazioni distribuite** che interagiscono scambiandosi messaggi in Internet.

Non in tutte le applicazioni un programma puramente client interagisce con un programma puramente server: nelle applicazioni per la condivisione di file **peer to peer**, l'applicazione agisce in alcuni casi da client (quando richiede un file) e in alcuni casi da server (quando li fornisce).

1.2.2 Internet come infrastruttura per applicazioni

Distribuzione delle applicazioni su Internet: login remoti, posta elettronica, navigazione web, messaggistica in tempo reale.

Web non è altro che **una delle molte applicazioni distribuite** che usano il servizio di comunicazione fornito da Internet.

1.2.3 Protocolli

Tutte le attività in Internet che coinvolgono almeno due entità remote atte alla comunicazione sono gestite da un protocollo che definisce:

1. Formato e ordine dei messaggi scambiati tra due o più entità comunicanti;
2. Le azioni che hanno luogo a seguito della trasmissione e/o ricezione di un messaggio o di altri enti

Tra i più importanti:

TCP (Transfer Control Protocol, protocollo di controllo della trasmissione).

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

IP (Internet Protocol, protocollo Internet): specifica il formato dei pacchetti che sono scambiati tra router e tra terminali.

HTTP (HyperText Trasmission Protocol).

I browser implementano il lato client dell'**HTTP**: per questo spesso vengono anche indicati semplicemente come **client**. Esempi di browser sono Netscape Communicator, Safari, Mozilla, ecc …

Spesso a fungere da client sono programmi, quali ad esempio bot o crawler.

Quando ci si riferisce al client di un'applicazione web, molto spesso si usa il termine più generale di **user agent**.

Un **server web** funge da deposito di oggetti web.

Tra i server web abbiamo Apache, Microsoft Internet Information Server, Netscape Enterprise Server.

1.2.4 **Pagine Web**

La crescita del web è stata immediatamente molto veloce. Nel 1994, il **W**orld **W**ide **W**eb **W**orm (WWWW) usava un indice di riferimento a circa 110.000 pagine web e documenti accessibili dal web. Nel novembre 1997, i principali motori di ricerca dichiaravano di indicizzare da 2 anni (WebCrawler) a 100 milioni di documenti web (Search Engine Web). Oggi Google dichiara di indicizzare miliardi di pagine.

Pagina web è un termine piuttosto generico che indica un insieme di oggetti web il cui rendering viene fatto contemporaneamente perché strettamente collegato.

Un **oggetto web** è una risorsa indirizzabile da uno specifico Uniform Resource Locator o **URL**; ad esempio:

- Un file **HTML**;
- Un'immagine **JPEG** o **GIF**;
- Un applet Java;
- Un audio clip.

Molte pagine web consistono di un documento HTML principale e da veri oggettoi indirizzati dai corrispondenti **Uniform Resource Identifier**.

Un **URI** ha la forma *schema : dettagli*.

Il termine URI è più generale del termine **URL** perché comprende anche gli **Uniform Resource Names** o **URN**.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

La differenza tra URN e URL assomiglia a quella tra il nome di una persona e il suo indirizzo: un **URN** serve solo a identificare univocamente un oggetto. Al contrario, un **URL** potrebbe avere la forma: file:///home/username/books/RomeoAndJuliet.pdf che ci dice come arrivare a una copia di quella risorsa salvata in locale. Ne consegue che **URN** e **URL** danno informazioni complementari:

Un **URL**, quindi, è un **URI** che dà le seguenti informazioni:

1. Il nome del protocollo (nel nostro caso, **http**);
2. Il nome dell'host su cui si trovano gli oggetti;
3. Il path degli oggetti sull'host.

Esempi di URL:

- **http://www.unina.it/...**
- **http://wpage.unina.it/anna.corazza/**
- **https://www.docenti.unina.it/anna.corazza**

Esempi di pagina web: si consideri una pagina web che contiene cinque immagini JPEG: in totale la pagina è composta da 6 oggetti, ovvero 1 file HTML e 5 file JPEG. La pagina principale fa riferimento alle immagini tramite i loro **URL**.

Abbiamo già visto che un **browser** è prima di tutto un client **HTTP**. Le informazioni possono arrivare all'interno della risposta HTTP in vari formati, testuali, grafici, video, audio, multimediali.

Il browser è di tipo **user agent**.

Il concetto di **user agent** è del tutto generale e indica un'interfaccia tra l'utente e l'applicazione; per fare un altro esempio, si pensi alla posta elettronica, per la quale lo user agent è rappresentato dal lettore di posta.

Nel caso del web, altri tipi di user-agent sono **sistemi di information retrieval**, che si muovono nel web alla ricerca di informazioni e gli **interpreti VXML** (VoiceXML) che hanno le stesse funzionalità del browser ma interagiscono con l'utente usando solo la modalità vocale.

Il web può essere visto come un enorme sistema client/server, in cui tutti i client e i server coesistono contemporaneamente sulla medesima rete.

In ogni istante la connessione riguarda **un solo** server, anche se due interazioni successive possono riguardare due server anche lontanissimi tra loro.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

1.3. **Dal web statico al web dinamico**

Inizialmente il web era caratterizzato da un tipo di scambio informativo **monodirezionale**: il client inviava una richiesta ad un server, che in cambio inviava un file, visualizzato dal browser sul client.

La comunicazione rapidamente è diventata **bidirezionale**, permettendo al client di passare alcune informazioni al server. Ovviamente il protocollo HTTP deve dare supporto a questo scambio di informazione bidirezionale.

Viene quindi introdotta la **programmazione lato server**.

Alla base di tutta questa architettura sta il vincolo che la stessa informazione deve venir mostrata nello **stesso** modo da tutti i client, quindi da tutti i browser. La necessità di assicurare che l'aspetto dell'applicazione sia lo stesso su tutti i browser è uno dei grossi problemi dello sviluppo di applicazioni web.

Si parla a questo proposito di **portabilità** inter-browser.

Ogni browser può implementare una resa diversa per gli stessi file di ingresso. Esempi di questi standard sono l'XHTML e, più recente, l'HTML5.

I primi browser erano molto primitivi: pensati per semplici ipertesti, non erano in grado di supportare nessuna interattività. Risultavano quindi **insufficienti**. In compenso ne giovava la **sicurezza**, perché i browser non potevano eseguire nulla, neppure codice malevolo o pericoloso per eventuali guasti in esso contenuti.

Sono stati introdotti nuovi strumenti:

- **Fogli di stile**: vi è presente il CSS e l'XSL;
- **Programmazione lato client**: incluse le applet Java, ma soprattutto JavaScript e AJAX.

L'**interattività** fornita dall'architettura server-browser iniziale era tutta a **carico del server**. Il server non faceva che produrre **pagine statiche** che il browser interpretava e visualizzava.

Il primo meccanismo introdotto per raccogliere ed elaborare questa informazione lato server è il **Common Gateway Interface** (CGI) messo a disposizione da tutti i server web.

In linea di principio praticamente qualsiasi operazione può essere implementata col meccanismo del CGI.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

I problemi legati a questo meccanismo sono:

- Manutenibilità;
- Tempo di risposta, che dipende da:
    1. Dimensione dei dati trasferiti;
    2. Carico del server;
    3. Carico della rete.

La **programmazione lato client** rappresenta una soluzione a questo problema di efficienza.

Con la programmazione lato client, il browser esegue parte del lavoro, rendendo possibile una maggiore interattività ed efficienza.

La programmazione lato client è sostanzialmente programmazione, ma all'interno del browser, che costituisce l'ambiente ristretto all'interno del quale vengono eseguite tutte le elaborazioni. Uno dei modi di realizzare la programmazione lato cliente è attraverso il **plug-in**.

Basta scaricare il plug-in **una volta sola**.

I plug-in sono utili per fornire al browser funzionalità veloci e potenti, il loro valore sta nella possibilità per il programmatore esperto di sviluppare un nuovo linguaggio e aggiungerlo al browser senza dover chiedere il permesso al produttore del browser.

L'effetto dell'introduzione dei plug-in fu un'esplosione dei **linguaggi di scripting** per la programmazione lato client. Un linguaggio di scripting permette di inserire il codice direttamente nella pagina HTML.

Vantaggi dei linguaggi di scripting sono:

- Di facile uso e comprensione;
- Veloci da caricare (testo nella pagina HTML);

Svantaggio:

- Codice esposto e facile da copiare;

Tra i linguaggi di scripting usati nei browser web sono usualmente progettati per risolvere problemi specifici, in particolare per la creazione di interfacce grafiche (GUI = Graphical User Interface) più ricche e interattive.

Tra i linguaggi di scripting più diffusi: **JavaScript, VBScript,** e **Tcl/Tk**, derivato dal popolare linguaggio inter-piattaforma per lo sviluppo di GUI.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Java si applica alla programmazione lato client attraverso le applet e Java Web Start, un modo relativamente recente di distribuire programmi standalone che non hanno bisogno di un browser in cui venire eseguiti.

Una **applet** è un programma che viene eseguito nel browser: viene scaricata come un qualsiasi oggetto web all'interno della pagine e, quando attivata, manda in esecuzione il programma. In questo modo il software viene distribuito dal server al client esattamente quando ce n'è bisogno, e non prima, permettendo di fornire sempre la versione più aggiornata, senza bisogno di reinstallazioni.

Data la caratteristica di Java di essere compilato in un bytecode, non occorre creare diverse versioni del programma adatte alle diverse piattaforme, ma lo stesso bytecode può venir eseguito da tutti i browser provvisti di un interprete Java.

Al contrario dei linguaggi di scripting, una applet Java viene scaricata nella sua forma compilata (bytecode): anche se non è immediatamente manipolabile, esso può essere decompilato abbastanza facilmente.

1.3.1 **Programmazione lato server**

Le richieste possono semplicemente riguardare degli oggetti web, quali pagine HTML, immagini grafiche, applet Java, script, ecc…

In molti casi le richieste sono più complicate e spesso coinvolgono l'accesso ad una **base di dati**:

- È possibile che la richiesta sia già stata elaborata dal lato client
- Comunque, viene inviata al server che la elabora e inoltra alla base di dati
- Sempre il server prepara la risposta della base di dati in una pagina web che rimanda al client.

Le elaborazioni dal lato server sono state implementate tradizionalmente con linguaggi come Perl, Python, C e C++, o con programmi CGI.

Approcci più recenti includono **servlets** e **portlets**, su server basati su Java. Notiamo che un vantaggio è quello di non dover dipendere dalle funzionalità del browser.

1.4. **Internet versus intranet**

Nello sviluppo di applicazioni web occorre prestare molta attenzione al fatto che il codice sviluppato per il client possa venir eseguito su piattaforme molto diverse senza guasti. Quando invece si sviluppano applicazioni per una intranet, si può contare su un ambiente molto più controllato, in cui il tipo di piattaforme coinvolte

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

sia noto a priori. Inoltre, in un'intranet, spesso esiste del (prezioso) codice legacy sviluppato per applicazioni tradizionali da portare su web.

Gli aggiornamenti del codice possono venir fatti via browser, e questo è un altro vantaggio delle applicazioni web anche su intranet.

1.5. **La sicurezza sul WWW**

Scaricare ed eseguire automaticamente programmi su Internet sono operazioni che presentano il rischio di diffusione di virus. Gli oggetti che si scaricano da web possono essere di vari tipi, tra i quali:

- File GIF, che non sono pericolosi;
- Codice di script, che di solito hanno grossi limiti su quello che possono fare;
- Codice Java compilato, la cui sicurezza è assicurata dal fatto che le applet vengono eseguite in una specie di "scatola di sabbia" (sandbox), al di fuori della quale non possono né accedere alla memoria né scrivere su disco;
- Componenti ActiveX, che sono estremamente pericolosi, perché non ci sono limiti a quello che possono fare.

Le **firme digitali** possono rappresentare una soluzione, nel senso che certificano chi è l'autore. Non rappresentano però una soluzione per i gusti non intenzionali e comunque, il tempo intercorso tra quando il codice viene scaricato e quando ci si accorge del danno, può rendere impossibile rintracciare l'autore.

Il web è vulnerabile anche agli attacchi **contro i server**.

In caso di attacco ad un server web usato dalle aziende, si corrono gravi rischi sia **economici** che in **termini di reputazione**.

Colpendo un server web può essere possibile ottenere l'accesso a dati e sistemi privati.

Gli attacchi possono essere:

- **Passivi:** tra cui, intercettazione del traffico di rete tra un browser e un server, accesso a informazioni riservate;
- **Attivi:** tra cui, la simulazione di altri utenti, la modifica dei messaggi in transito fra client e server e l'alterazione delle informazioni contenute in un sito web.

e possono riguardare:

- Il server web;
- Il browser web;
- Il traffico di rete in transito tra server e browser.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

2) **IL PROTOCOLLO HTTP**

2.1. **Il protocollo HTTP (HyperText Transfer Protocol)**

Il protocollo HTTP rappresenta un po' il cuore del web, che di fatto è un'applicazione client-server. Una caratteristica importantissima del protocollo HTTP è di essere **senza stato:** questo significa che il protocollo non prevede che né il server né il client mantengano informazioni sullo stato della comunicazione. Ovviamente questo significa che se un'applicazione web ha bisogno di mantenere delle informazioni sullo stato della comunicazione, deve implementare meccanismi che lo facciano, non essendo sufficiente appoggiarsi al protocollo.

HTTP deve definire:

- Tipo dei messaggi scambiati (richiesta e risposta);
- Sintassi dei vari tipi di messaggio;
- Semantica dei diversi campi;
- Regole che determinano quando e come un processo invia messaggi o risponde a messaggi.

HTTP prevede esclusivamente due tipi di messaggio, entrambi in formato **testuale**:

1. **Richiesta** dal client verso il server;
2. **Risposta** dal server verso il client;

I tipi di messaggio sono composti da:

1. Una prima riga, **request line** nella richiesta e **status line** nella risposta;
2. Un certo numero di **righe di intestazione** (anche nessuna);
3. Una riga vuota;
4. Il **body** del messaggio.

Due tipi di **messaggi** HTTP: di richiesta e di risposta.

Ecco un esempio di messaggio di **richiesta:**

GET/somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/4.0
Accept-language: fr

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

La prima riga viene chiamata **request line** e ha tre campi:

1. **Metodo**: in HTTP/1.0:
   - GET: richiede un oggetto;
   - POST: invia informazione;
   - HEAD: come GET, ma restituisce solo intestazione; usato per debugging;

In HTTP/1.1, abbiamo anche:

   - PUT: carica un oggetto;
   - DELETE: cancella un oggetto;
   - Altri (TRACE, CONNECT)
2. **URL**: oggetto richiesto;
3. **Versione HTTP.**

Seguono le linee di intestazione:

1. Host;
2. Connection;
3. User-agent;
4. Molti altri, e HTTP/1.1 ne ha introdotti ancora di nuovi

Segue il **body**.

Nello scambio di informazione bidirezionale, l'informazione passa dal client al server all'interno della richiesta HTTP in un modo che dipende dal metodo HTTP adottato. Se si usa il GET, l'informazione viene passata come parametri alla fine dell'URL. Se invece si usa un POST, l'informazione viene passata nel body del messaggio.

HTTP di **risposta**, contiene dunque l'informazione che il browser visualizzerà:

HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html

dati … dati …

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

La prima riga viene detta **status line** e ha tre campi:

- Versione del protocollo
- Codice di stato:
    - 200 OK;
    - 301 Moved Permanently: il nuovo URL sta nell'intestazione in Location:, in modo che il client possa accedervi automaticamente;
    - 400 Bad Request;
    - 404 Not Found;
    - 505 HTTP Version Not Supported;
- Corrispondente messaggio di stato

1xx. (Informativo): la richiesta è stata ricevuta, l'elaborazione continua;
2xx. (Successo): la richiesta è stata non solo ricevuta, ma anche compresa e accettata;
3xx. (Re direzione): sono necessarie ulteriori operazioni per portare a buon fine la richiesta;
4xx. (Errore lato client): la richiesta contiene errori sintattici e non può quindi venire accolta;
5xx. (Errore lato server): il server non è in grado di soddisfare una richiesta che appare corretta.

HTTP si occupa solo ed esclusivamente della comunicazione tra client e server Web. Occorre quindi che l'informazione sul tipo di file sia passata dal server al cliente: questo è il ruolo della linea di intestazione **Content-Type:**. Se assente, si intende **text/html**.

2.1.1. **Esempio: richiesta HTTP via telnet**

Si provi a fare una richiesta HTTP via telnet. Ad esempio, da una macchina UNIX:

```
telnet wpage.unina.it 80
```

apre una connessione TCP alla porta 80 dell'host telnet wpage.it 80

```
GET /anna.corazza HTTP/1.0
```

Occorre inserire due a capo per segnalare la fine dell'header.

corazza@desktop: ~$ telnet wpage.unina.it 80
Trying 192.132.34.19…
Connected to wpage.unina.it.
Escape character is '^]'.
GET /anna.corazza/index.html HTTP/1.0

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

HTTP/1.1 200 OK
Date: Tue, 04 Sep 2018 07:12:51 GMT
Server: Apache
Last-Modified: Sat, 04 Aug 2018 17:02:01 GMT
ETag: “cf5399-81c-b854400”
Accept-Ranges: bytes
Content-Length: 2076
Connection: close
Content-Type: text/html; charset = ISD-8859-1

<!DOCTYPE html PUBLIC “-//W3C//DTD HTML 4.01 \
Transitional//EN” “http://www.w3.org/TR/html4/loose.dtd”>

<html>

<head>
<title> Anna Corazza: home page</title>
…
</body>
</html>

Se sostituiamo il metodo GET con il metodo HEAD

corazza@desktop: ~$ telnet wpage.unina.it 80
Trying 192.132.34.19…
Connected to wpage.unina.it.
Escape character is ‘^]’.

HEAD /anna.corazza/index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 04 Sep 2018 07:18:18 GMT
Server: Apache
Last-Modified: Sat, 04 Aug 2018 17:02:01 GMT
ETag: “cf5399-81c-b85440”
Accept-Ranges: bytes
Content-Length: 2076
Connection: close
Content-Type: text/html; character=ISD-8859-1

Altri metodi HTTP/1.0: POST, HEAD… e HTTP/1.1: PUT, DELETE con HTTP/1.1 occorre mettere una riga di header con Host, anche vuoto:

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

corazza@desktop: ~$ telnet wpage.unina.it 80
Trying 192.132.34.19…
Connected to wpage.unina.it.
Escape character is '^]'.
HEAD /anna.corazza/index.html HTTP/1.1
Host:

HTTP/1.1 200 OK
Date: Tue, 04 Sep 2018 07:18:18 GMT
Server: Apache
Last-Modified: Sat, 04 Aug 2018 17:02:01 GMT
ETag: "cf5399-81c-b85440"
Accept-Ranges: bytes
Content-Length: 2076
Connection: close
Content-Type: text/html; character=ISD-8859-1

2.2.1. **Meccanismo di cache**

Invece di scaricare tutta la pagina ogni volta che accediamo al sito, ne conserveremo sempre l'ultima copia e la scaricheremo solo quando è necessario. Naturalmente se fosse il client a dover decidere quando la pagina è stata modificata, il meccanismo non funzionerebbe.

Perché un meccanismo di cache possa venir adottato proficuamente, occorre che il client sia in grado di controllare se la pagina richiesta dall'utente è stata modificata oppure se quella che è stata salvata in cache è ancora valida senza riscaricare la pagina.

HTTP prevede il campo **If-Modified-Since** da inserire nell'intestazione della richiesta ed il codice **304 Not Modified** da inserire nella status line della risposta. Potremmo quindi riaccedere al sito chiedendo di scaricare la pagina solo se è stata modificata.

2.2.2. **Session tracking: cookies e autenticazione**

L'adozione di strategie di **session tracking** è resa necessaria dal fatto che il protocollo HTTP è **senza stato**, nel senso che i messaggi HTTP non contengono informazioni sugli scambi precedenti tra client e server. Tali informazioni risultano però necessarie almeno in alcuni casi per lo svolgimento delle funzionalità richieste dall'utente.

Un **cookie** è un pezzo di testo che viene scambiato tra client e server secondo opportune modalità.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Esso contiene alcune informazioni essenziali:

1. Host e path dell'applicazione;
2. Expiration date;
3. Nome e valore del cookie;
4. Altro.

Ogni cookie viene sempre creato dal server, che lo passo al client nel campo **set-cookie** nell'intestazione della risposta HTTP. Il client **può** decidere di salvarlo.

Questo avviene se in un browser i cookies sono abilitati per quella particolare applicazione, univocamente determinata dall'host e dal path contenuti all'interno del cookie.

Ogni volta che lo user-agent prepara una richiesta per una certa applicazione identificata all'interno dell'host da un particolare path, inserisce all'interno dell'intestazione della richiesta tutti i cookie che ha memorizzato e che corrispondono a quella combinazione di host e path. Quindi se ripeto la stessa richiesta di prima, il browser aggiungerà all'header una riga al cookie.

A questo punto il server riceve il cookie e quindi evita di inserire nella risposta il campo Set-Cookie. Si noti che la scelta se inserire questo campo nell'intestazione della risposta non può venir eseguita automaticamente dal server, ma richiede **programmazione lato server**.

L'expiration date serve a consigliare allo user-agent quando cancellare il cookie. Se l'expiration date è negativo, allora lo user agent dovrebbe cancellare il cookie immediatamente.

Questo serve perché ogni cookie è univocamente determinato da host, path e nome. Se il server manda un cookie con la stessa combinazione di queste tre informazioni, allora il cookie viene sovrascritto. Se il server vuole che un cookie venga cancellato, ne invia quindi uno con lo stesso nome di quello da cancellare, in modo che quest'ultimo venga prima sovrascritto e immediatamente cancellato.

Per quanto riguarda l'**autenticazione**, una volta fatta richiesta, l'host risponderà con una richiesta di autenticazione tramite messaggio **401 Authorization Required**.

A questo punto lo user agent deve procurarsi le credenziali per accedere al sito. I browser di solito la prima volta producono una finestrella di pop-up in cui l'utente inserisce username e password. Al posto della finestrella di pop-up il browser può usare la finestra principale. L'utente inserisce le credenziali che in questo modo vengono messe a disposizione del browser, che non solo le inserisce nella richiesta

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

che sottopone al server, ma, in aggiunta a ciò, le salva, in modo da poterle introdurre in ogni successiva richiesta.

L'HTTP fornisce due meccanismi a sostegno del session tracking:

- Autenticazione: il codice 401 della status line e i campi per la gestione dell'autorizzazione;
- Cookies: i campi Set-Cookie e Cookie per lo cambio dei cookies.

L'HTTP fornisce anche un meccanismo che permette al client di impiegare il caching pur assicurando che gli oggetti passati al browser siano aggiornati:

- GET condizionato (**conditional GET**).

3) **HTML**

3.1. **Cos'è HTML?**

Si tratta di un **linguaggio di etichettatura** per ipertesti: HTML sta infatti per HyperText Markup Language.

La funzione dell'annotazione è quella di dare alcune indicazioni su come il testo andrà formattato.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>Qui va il titolo del documento</title>
</head>
<body>
qui va il contenuto del documento
</body>
</html>
```

La prima riga del documento dichiara quale versione di HTML si adotta. In questo caso si tratta di HTML4.

Il documento HTML vero e proprio è poi composto da un insieme di **elementi**: un elemento è una parte di documento compresa tra l'apertura di un'**etichetta** (ad esempio, <head>) e la relativa chiusura (</head>), etichette comprese. L'etichetta (o tag) rappresenta il **nome dell'elemento**.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Non bisogna confondere il concetto di **elemento** con quello di **etichetta**. Infatti, l'elemento è una parte di documento, mentre l'etichetta è semplicemente un nome. In un documento possono esserci più elementi con la stessa etichetta. La stessa considerazione ovviamente vale anche per l'XML.

Va tenuto presente che HTML non dà indicazioni precise sull'aspetto grafico, ma solo sulla funzione delle diverse parti di test. Per definire in dettaglio gli aspetti della rappresentazione grafica, ed eventualmente ai diversi media su cui si ha il rendering del documento, occorre introdurre dei **fogli di stile** e in particolare un **Cascading Style Sheet** o **CSS**. In altre parole:

- HTML descrive la **struttura** del documento;
- CSS descrive invece il **layout** grafico o sonoro del documento stesso.

Sottolineiamo che il termine rendering include sia la resa grafica che quella sonora.

Ovviamente HTML deve prevedere la possibilità di introdurre dei **link:**

This is a link to Peter's page.

HTML, però oltre alle indicazioni sulla struttura del documento, prevede anche delle etichette per **interagire** con l'utente. Infatti, l'utente può dare origine a degli **eventi** a cui verranno associate diverse azioni. (Esempio: form/bottoni).

FORM = rappresentano un modo per passare parametri al server

```
<FORM action="http://somesite.com/prog/adduser" method="post">
  <P>
    <LABEL for="firstname">First name: </LABEL>Appunti di Tecnologie Web 37 <INPUT type="text" id="firstname">
    <BR>
    <LABEL for="lastname">Last name: </LABEL>
    <INPUT type="text" id="lastname">
    <BR>
    <LABEL for="email">email: </LABEL>
    <INPUT type="text" id="email">
    <BR>
    <INPUT type="radio" name="sex" value="Male"> Male <BR>
    <INPUT type="radio" name="sex" value="Female"> Female <BR>
    <INPUT type="submit" value="Send">
    <INPUT type="reset">
  </P>
</FORM>
```

3.1.1. **Differenza tra gli attributi name e id**

L'attributo **name** determina il nome del parametro che viene passato al server. Ci potrebbero essere più parametri con lo stesso **name**. Al contrario il valore di **id** deve essere univoco: ne consegue che al di fuori delle form, conviene sempre usare **id** al posto di **name**, che infatti è biasimato.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

3.2. **XML e XHTML**

XML sta per e**X**tensible **M**arkup **L**anguage: si tratta dunque di un **linguaggio di markup** come HTML, ma con la caratteristica di essere estendibile, nel senso che le etichette non sono prefissate, ma possono essere scelte dall'utente. Questo perché è stato progetto per **descrivere dati**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<corso>
<crediti>6</crediti>
<anno>4</anno>
<docente>Anna Corazza</docente>
<nome>Tecnologie Web</nome>
</corso>
```

XHTML è una variante dell'HTML, che segue la sintassi dell'eXtensible Markup Language o XML. Le etichette sono le stesse dell'HTML, ma la sintassi è leggermente diversa. Il fatto che un documento XHTML sia un documento XML, ha il vantaggio di permettere l'applicazione di tutti gli strumenti sviluppati per XML.

3.3. **HTML5**

HTML5 introduce delle modifiche piuttosto importanti, che pongono problemi di compatibilità con le precedenti versioni di HTML. Ovviamente gli user agents devono essere in grado di gestire sia HTML5 che le precedenti versioni.

Mentre gli sviluppatori di applicazioni web, chiamati **autori** nelle specifiche, devono evitare di usare questi elementi, gli **user agents** devono tuttavia essere in grado di gestirli in modo corretto. I requisiti vengono divisi in due parti:

1. **Requisiti per gli autori;**
2. **Requisiti per gli user agents.**

3.3.2. **Motivazioni di HTML5 e principi di sviluppo**

1. Lo sviluppo di pagine di contenuti da pubblicare in web ormai non può prescindere da un gruppo di strumenti che si integrano strettamente l'uno con l'altro:
    a. **HTML** per l'annotazione degli ipertesti; necessita di separare le indicazioni di formattazione, da quelle più legate ai contenuti di testo. Vantaggi: portabilità e manutenibilità;
    b. **CSS** (Cascading Style Sheets) si occupa più precisamente della formattazione nel rendering dei contenuti.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

c. **DOM** (Documento Object Model) dà indicazioni su come il documento HTML viene rappresentato in memoria mediante delle API. Si tratta di un albero in cui ogni nodo corrisponde ad un elemento del documento.
d. **JavaScript**, un linguaggio di scripting che permette di elaborare la DOM per eseguire operazioni lato client.

Occorre quindi introdurre in HTML5 nuove features che siano basate non solo su HTML, ma anche su CSS, DOM e JavaScript.

2. Ridurre la necessità di plugins esterni, come Flash;
3. Migliorare la gestione degli errori;
4. Laddove possibile, sostituire l'uso di un linguaggio di scripting con un'adeguata annotazione (markup);
5. Indipendenza dal device;
6. Il processo di sviluppo delle specifiche deve avvenire in modo trasparente verso il pubblico.

Con il termine **semantico** ci si riferisce alla descrizione della struttura e non dei contenuti del documento. Viene quindi contrapposto alla presentazione del documento. Vengono quindi completamente rimosse tutta una serie di etichette tese a descrivere la formattazione del documento, oltre a elementi ormai sostituiti da altri, come ad esempio applet.

Sono state quindi inserite delle nuove etichette, che si riferiscono alla struttura del documento, tra cui section, article, footer, progress, nav.

3.3.3. **Doppia sintassi**

HTML5 può seguire sia la sintassi HTML che quella XML:

- test/html: mine
- XML: il documento va definito come un documento XML in cui tutti i documenti vanno messi in un apposito namespace
  <html xmlns=http://www.w3.org/1999/xhtml> ← namespace

3.3.4. **Memorizzazione (storage)**

Un'importante caratteristica di HTML5 riguarda la possibilità di memorizzare dati localmente all'interno del browser, in alternativa ai cookies, sempre come coppia di stringhe nome=valore. Questi dati non vengono inseriti in ogni richiesta HTTP inoltrata al server, ma solo quando risulta necessario farlo. Senza tuttavia pesare sulle prestazioni del sito.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

3.4.1. **Same-origin policy**

Gli user agent eseguono azioni suggerite loro dai contenuti che scaricano dai diversi siti. Alcuni siti possono provvedere dei contenuti malevoli, occorre che lo user agent applichi delle politiche per salvaguardare le informazioni di cui è responsabile, sia proprie che altrui.

Ad uno user agent quasi sempre viene richiesto di mandare in esecuzione script prodotti dal sito che sta visitando. Ovviamente occorre cautelarsi.

Per affrontare questo problema, gli user agent usano il concetto di **origine** per implementare la cosiddetta **same-origin policy.** Il principio seguito è che i contenuti possono interagire senza restrizione con tutti i contenuti provenienti dalla stessa origine, mentre questo non può avvenire tra contenuti di origine diversa.

Due URI sono considerate appartenere alla stessa origine se hanno uguali **schema**, **host** e **porta**.

3.4.2. **Mancata validazione dell'input dell'utente; cross-site scripting (XSS); SQL injection**

Occorre quindi prevedere filtri per la validazione dei dati in ingresso, ma è importantissimo che questi filtri siano basati sull'enumerazione dei casi favorevoli (in inglese, **whitelist-base**), bloccando tutto ciò che non sia esplicitamente previsto.

Si manda in esecuzione uno script scelto dall'attaccante che può portare a termine azioni ostili, nei limiti delle azioni offerte dal sito. Un attacco di questo tipo viene chiamato **cross-site scripting attack**.

3.4.3. **Cross-site request forgey (CSRF)**

In un sito in cui si permette all'utente di usare la sottomissione di form per portare a termine delle azioni a nome dell'utente, come inviare messaggi, compiere degli acquisti, richiedere documenti, è della massima importanza verificare che l'utente abbia fatto la richiesta intenzionalmente e non perché tratto in inganno. L'origine di questo problema risiede nel fatto che HTML permette all'utente di sottomettere le form anche a origini diverse da quelle dell'applicazione.

3.4.4. **Clickjacking**

Supponiamo di avere una pagina web in cui siano previste delle azioni che l'utente può non voler intraprendere: in questo caso bisogna garantire che non sia possibile che l'utente le scelga senza averne coscienza.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

4) CGI – Common Gateway Inteface

Programmazione lato server: Common Gateway Interface (CGI).

4.2. **Il meccanismo del CGI**

Si tratta di un meccanismo per far eseguire al server Web dei programmi (o degli script). Tra le funzionalità del server web ci sarà quella di permettere di impostare la configurazione in modo che i programmi CGI si possano eseguire. Poiché il meccanismo del CGI permette all'utente di richiedere l'esecuzione di programmi sul server, per tutelare la sicurezza del server stesso occorre assicurarsi che tali programmi non abbiano possibilità di nuocere.

Il meccanismo del CGI viene usato molto per riutilizzare sistemi legacy: essi vengono "avvolti" in un wrapper e tramite CGI resi accessibili su Web.

Un'altra applicazione tipica ha lo scopo di costruire sistemi di accesso a basi di dati, anche se ormai altri sistemi hanno un tale supporto da essere molto più competitivi.

L'accesso ad una base di dati è un esempio tipico di applicazione web dinamica e il suo funzionamento si può scomporre nelle seguenti fasi:

1. Il **browser** raccoglie i parametri, ad esempio usando un form HTML, ma non solo e li introduce nella **richiesta HTTP** che passa al server;
2. Il **server** usa questi parametri per costruire la query di accesso al **database**;
3. Il **server** organizza i risultati della query in una tabella che passa nella **risposta HTTP** al **browser**.

Va sottolineato che il CGI, pur essendo un meccanismo poco sofisticato, è molto flessibile e viene quindi utilizzato in molti contesti diversi.

Si consideri infatti il caso in cui se non c'è un programma CGI in grado di gestire la richiesta: una risposta deve necessariamente essere prodotta e questo non può che farlo il server web. Se invece le cose vanno bene e il programma CGI viene messo in esecuzione, **il server web produce solo la status line, mentre il programma CGI produce il resto della risposta.**

L'idea generale è che quando lo user agent manda al server una richiesta HTTP per una risorsa relativa ad un programma sul CGI, il corrispondente programma viene mandato in esecuzione sul server e i risultati vengono ripassati allo user agent mediante la risposta HTTP.

Occorre impostare il server in modo che possa riconoscere quali richieste vanno gestite con CGI.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

4.2.1. **Input al programma CGI**

La modalità con cui i parametri vengono passati al programma dipende dal tipo di metodo utilizzato. Vengono utilizzati lo **stdin** e le **variabili d'ambiente**,

1. Richiesta HTTP di tipo **GET**: per il passaggio dei parametri viene utilizzata la variabile d'ambiente QUERY_STRING in cui viene inserito tutto ciò segue il ? nell'URL; ovviamente la lunghezza è limitata;
2. Richiesta HTTP di tipo **POST**: tutto quello che si trova nel body della richiesta viene passato al processo CGI sullo **stdin**; non viene terminato con un **EOF**, e quindi occorre conoscerne la lunghezza: a tale scopo viene usata la variabile d'ambiente **CONTENT-LENGHT.**

4.2.2. **Output del programma CGI**

La risposta HTTP da inoltrare allo user agent viene preparato utilizzando lo **stdout** del processo CGI, a cui il server **esclusivamente** la riga di stato.

- Lo user agent invia una richiesta HTTP al server;
- Il server web la riconosce come programma Common Gateway Interface;
- Il server web manda in esecuzione il programma;
- Se l'esecuzione del programma produce dei risultati, questi vanno organizzati sullo **stdout.**
- il server Web prepara la risposta HTTP da inviare il cliente includendo:
    - status line;
    - output del programma.

Se viene prodotto un file (documento, audio, video,…) da passare al browser, occorre un'intestazione che dà il tipo MIME del file: Content-type, che può essere **text/html** o **tt text/plain**.

Perché lo user agent sappia come trattare il documento che gli viene passato, occorre dichiarare nell'intestazione della risposta HTTP il tipo del documento e se si tratta di un riferimento. Il meccanismo CGI assegna questa responsabilità allo sviluppo del programma CGI.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Esempio scipt.cgi (date. Cgi)

```
#!/bin/csh -f

echo "Content-type: text/html"
echo ""

echo "<HTML>"
echo "la data di oggi: "
date
echo "</HTML>"
```

Altro esempio che restituisce le variabili d'ambiente:

```
#!/bin/sh
# cgi-test.sh

echo "Content-type: text/html"
echo
echo "<HTML>"
echo "<HEAD>"
echo "<TITLE>Test CGI</TITLE>"
echo "</HEAD>"
echo "<BODY>\n";
echo "<H1>Test CGI</H1>"
echo "<PRE>"
echo "N. argomenti = $#"
echo "Argomenti = $*"
echo
echo "SERVER_SOFTWARE = $SERVER_SOFTWARE"
echo "SERVER_NAME = $SERVER_NAME"
echo "GATEWAY_INTERFACE = $GATEWAY_INTERFACE"
```

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

```
echo "SERVER_PROTOCOL = $SERVER_PROTOCOL"
echo "SERVER_PORT = $SERVER_PORT"
echo "SERVER_ADMIN = $SERVER_ADMIN"
echo "REQUEST_METHOD = $REQUEST_METHOD"
echo "HTTP_ACCEPT = $HTTP_ACCEPT"
echo "HTTP_USER_AGENT = $HTTP_USER_AGENT"
echo "HTTP_CONNECTION = $HTTP_CONNECTION"
echo "PATH_INFO = $PATH_INFO"
echo "PATH_TRANSLATED = $PATH_TRANSLATED"
echo "SCRIPT_NAME = $SCRIPT_NAME"
echo "QUERY_STRING = $QUERY_STRING"
echo "REMOTE_HOST = $REMOTE_HOST"
echo "REMOTE_ADDR = $REMOTE_ADDR"
echo "REMOTE_USER = $REMOTE_USER"
echo "AUTH_TYPE = $AUTH_TYPE"
echo "CONTENT_TYPE = $CONTENT_TYPE"
echo "CONTENT_LENGTH = $CONTENT_LENGTH"
echo
echo "Standard input:"
cat -
echo "</PRE>"
echo "</BODY>"
echo "</HTML>"
```

Un programma CGI può venire scritto in qualsiasi linguaggio di programmazione.

Un processo CGI deve sottostare a delle restrizioni. La prima è che i programmi CGI devono risiedere in una directory particolare, in modo che il server Web sappia che sono programmi da mandare in esecuzione non semplici.

Ogni volta che un client richiede, attraverso una richiesta HTTP, la risorsa corrispondente al programma CGI, il serve lo manda in esecuzione. Se arrivano più richieste per la stessa risorsa, il programma viene mandato in esecuzione più volte, ottenendo così più processi che vengono eseguiti contemporaneamente,

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

mantenendo tuttavia l'informazione che permette di accoppiare il processo con la relativa richiesta HTTP, perché ovviamente va accoppiata la risposta.

"**Le servlet permettono di condividere delle informazioni** sia all'interno di un'applicazione che col server mentre nelle CGI, l'unico ambiente di definizione (o scope) previsto è quello di una richiesta e per condividere informazioni tra due diverse richieste serve salvarle in modo persistente.

Un altro problema del CGI è che non dà modo ai programmi CGI di interagire direttamente col server o comunque di usare le funzionalità del server dopo che il programma è entrato in esecuzione; **nelle servlet, ad ogni richiesta corrisponde un nuovo thread all'interno del servlet engine**: thread diversi possono accedere a variabili comuni, e questo permette di condividere informazioni tra richieste diverse alla stessa servlet. Ovviamente ci possono essere, e vanno di conseguenza gestiti, problemi di sincronizzazione tra richieste/thread diversi."

Il meccanismo del CGI usa variabili di ambiente per passare i parametri al programma, e lo **stdin** per passare il contenuto del body della richiesta.

Il CGI prima di ripassare la stringa al programma, deve ritrasformarla: si parla al proposito di parsing.

Si ricordi che l'**output prodotto dal programma viene inserito nella risposta HTTP da rispedire al client: occorre quindi che sia opportunatamente formattato**.

4.1.1. **Considerazioni riguardo alla sicurezza**

Ogni vola che un programma interagisce con un client attraverso la rete, occorre valutare il richiesto che il client possa ottenere controllo per cui non è autorizzato attraverso un attacco a quel programma.

Comportamenti suggeriti sono:

- Evitare l'istruzione **eval**;
- Non fidarsi a lasciar far nulla al client;
- Prestare attenzione a **poppen** e **system;**
- Chiudere tutti gli include sul server.

Brevemente lo intendiamo come un meccanismo che esegue dei programmi o script su un server web. Questi programmi non sono avviati in modo automatico sul server; infatti, il server deve essere abilitato (a patto che sappia cosa stia facendo) ad eseguire programmi CGI sulla macchina al fine di evitare spiacevoli inconvenienti.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Questo meccanismo viene usato principalmente per ridare vita nel web a sistemi legacy, rendendoli accessibili su questo; d'altronde è anche utilizzato per varie mansioni in quanto è un meccanismo molto flessibile. Il funzionamento generale lo si può riassumere così: una volta che lo user agent manda al server una richiesta HTTP contenente un'elaborazione riservata al programma CGI, il server riconosce questa richiesta ed "aziona" il programma CGI e, una volta ottenuta la risposta, la incorpora nella risposta HTTP.

5) JAVA

5.1. **Java per applicazioni Web**

Java è stato progettato fin dall'inizio come un linguaggio con caratteristiche di sicurezza e buona programmazione che lo rendono particolarmente adatto alle applicazioni web. E' utile il fatto che sia un linguaggio **orientato agli oggetti**, che abbia una gestione automatica del **garbage collector** della JVM, il suo essere **indipendente dalla piattaforma** e per le sue caratteristiche di **sicurezza** (la maggior parte delle azioni di attacco ad un sistema non possono essere descritte come un programma Java).

In particolare, per le questioni di sicurezza i programmi Java **non possono richiamare funzioni globali, ne ottenere l'accesso a risorse del sistema in modo arbitrario**: la macchina virtuale Java è in grado di esercitare un grado di controllo impossibile ad altri sistemi.

Java è più robusto, nel senso che offre un **miglior controllo degli errori**. La maggior parte degli errori sono legati a due aspetti: **gestione della memoria** e **condizioni eccezionali**.

La sintassi di Java è molto rigida per quel che riguarda la gestione dei tipi e le dichiarazioni: questa da la **possibilità di rilevare molti errori giù durante la compilazione.**

Un **buon supporto al multi-threads** può essere nell'implementazione di interfacce sofisticate e nella gestione di richieste concorrenti.

Il programma Java viene compilato in un **bytecode** che viene eseguito dalla macchina virtuale (che funge da interprete). In questo modo si ottiene un buon compromesso tra linguaggi compilati e linguaggi interpretati:

- È più **portabile** di un linguaggio compilato;
- È più **efficiente** di un linguaggio interpretato;

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

- Ha caratteristiche che un linguaggio interpretato di solito non ha (ad esempio è fortemente tipato).

È più **facile** di molti altri linguaggi.

Ha un'**ampia disponibilità di librerie**, in particolare per la rete le classi e le interfacce del pacchetto java.net per i socket, dei 5 pacchetti java.rmi per la Remote Method Invocation.

La Java Virtual Machine è una macchina astratta e non vengono fatte ipotesi sulla particolare tecnologia in cui viene implementata né sulla piattaforma ospite. Un file class contiene istruzioni per la Java Virtual Machine, una tabella di simboli e altre informazioni ausiliarie.

Una piattaforma Java è composta da:

- Linguaggio Java;
- Java Virtual Machine;
- Application Programming Interfaces (API libraries).

Comunque l'aspetto più importante di Java è la **sicurezza**. Essa dipende da:

- Il progetto del linguaggio come sicuro e di facile uso;
- Verifica del bytecode prima dell'esecuzione;
- Durante l'esecuzione:
    - Le classi vengono caricate e ne viene fatto il linking;
    - Opzionale: generazione del codice macchina e ottimizzazione dinamica;
    - Esecuzione del programma: durante questo processo, il class loader definisce un namespace locale, in modo che codice non sicuro non possa interferire con l'esecuzione di altri programmi Java.
- La mediazione da parte della Java Virtual Machine mediante una classe security manager per l'accesso alle risorse di sistema.

È sempre importante trovare il giusto compromesso tra:

- Sicurezza;
- Usabilità;
- Prestazioni.

5.1.1. Caratteristiche di sicurezza a livello del linguaggio

Una delle caratteristiche di altri linguaggi che rende impossibili questi controlli è l'**aritmetica dei puntatori**, che in Java è invece esclusa.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Accessi non protetti alla memoria rendono possibili alcuni degli attacchi più frequenti alla sicurezza delle applicazioni.

Infatti, se ai programmi è permessa ogni manipolazione degli indirizzi attraverso puntatori senza limitazioni specifiche imposte dal tipo, non può essere applicato a nessun concetto di **dato privato**.

I controlli sull'accesso alla memoria vanno implementati sia a livello di linguaggi, non permettendo quelle caratteristiche che li renderebbero possibili, che durante l'esecuzione, perché non tutti i controlli necessari possono essere implementati staticamente.

Ovviamente vietare l'aritmetica dei puntatori non equivale a non supportare una qualche forma di puntatori, che in Java sono chiamati **references.**

Oltre ad abolire l'aritmetica dei puntatori, le specifiche di Java a livello di linguaggio definiscono chiaramente il comportamento di **variabili non inizializzate.**

Le variabili:

- Sullo **heap** vengono automaticamente inizializzate; ne consegue che le variabili di classi e istanze non possono avere valori non definiti;
- Stack;
- Sullo **stack** non vengono automaticamente inizializzate e quindi tale operazione deve avvenire in modo esplicito prima dell'uso pena un errore di compilazione.

Un altro aspetto di Java che contribuisce alla robustezza e quindi alla safety del codice è la **garbage collection** automatica. Per **garbage collection** si intende la capacità della macchina virtuale di liberare automaticamente la memoria che non viene più indirizzata.

Ovviamente la garbage collection riguarda solo la heap.

Infine, tra gli aspetti che rendono più difficili gli errori annidati nel codice e quindi più affidabile il codice Java, va riportato il controllo dei tipi di forte fatto durante la compilazione, che esclude operazioni di cast non legali.

Questo controllo evita che un blocco di memoria venga interpretato in modo diverso da quello che è, col rischio di dare accesso palesemente illegale a zone di memoria.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

5.1.2. **Caratteristiche di sicurezza a livello di macchina virtuale**

La macchina virtuale Java usa una **sandbox** per mandare in esecuzione programmi Java:

- Per verificare tutte le classi che entrano;
- Per operare numerosi controlli a runtime che evitano che i programmi Java compiano azioni non valide;
- Per alzare una barriera attorno all'ambiente di runtime e controllare tutti gli accessi all'esterno della sandbox.

Non tutti i controlli possono essere fatti staticamente (durante la compilazione).

Il peggio che può accadere è una condizione di denial of service, in cui il programma consuma tutte le risorse della CPU.

Il "linguaggio macchina" della macchina virtuale Java è il bytecode, definito dalle specifiche della macchina virtuale. Un compilatore converte in bytecode (un file .class) il file sorgente, che però può anche non essere codice Java.

Se ne evince che **non è sufficiente implementare costrutti sicuri del linguaggio**, perché il sistema potrebbe comunque essere attaccato dal del bytecode ostile generato da compilatori Java modificati, da compilatori che partono da linguaggi meno sicuri, o addirittura a mano: è necessario implementare anche dei controlli sull'ambiente di esecuzione.

Ogni volta che una classe viene caricata, il bytecode viene verificato.

La verifica del bytecode caricato è compito del ClassLoader, che solo al termine della verifica crea le classi vere e proprie. Il ClassLoader non cerca mai di caricare le classi di java.* da remoto: in questo modo si evita che esse possano venir sostituite con classi modificate in cui i meccanismi di sicurezza siano stati in qualche modo manomessi.

A classi provenienti da host diversi non è permesso di comunicare all'interno della macchina viruale Java, in modo da evitare che programmi non necessariamente affidabili possano ottenere informazioni da programmi considerati affidabili.

Una volta che le classi sono state caricate, la macchina virtuale compie i controlli necessari a tempo di esecuzione. Sono necessari controlli durante l'esecuzione sia per quel che riguarda i tipi nelle istruzioni di assegnamento che per quel che riguarda i limiti di vettori.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

In Java i riferimenti agli oggetti includono informazioni complete sulla classe di cui l'oggetto è un'istanza: questo rende possibile il controllo della compatibilità tra tipi a runtime.. se il controllo fallisce, viene sollevata un'eccezione.

Ogni macchina virtuale Java in esecuzione ha al più un SecurityManager installato, responsabile per la gestione delle politiche di sicurezza. SecurityManager è una classe del pacchetto java.lang: se ne può quindi definire una sottoclasse e usare il metodo System.setSecurityManager() per stabilire un security manager personalizzato.

Una volta che un manager è stato installato, ogni tentativo di sostituirlo genera un errore, quindi nessuno può variare questa funzione in modo ostile, rimpiazzandola.

Attraverso le sue politiche è possibile allentare o restringere i vincoli sulle operazioni che possono essere eseguite da un programma Java: ogni tentativo di eseguire un'operazione vietata solleva un Security Exception.

5.2. **Applets**

Si tratta di un esempio di strumento per la programmazione **lato client** basato su Java. Cruciali le caratteristiche di:

- **Portabilità,** visto che lo sviluppatore non sa praticamente nulla della piattaforma su cui l'applet verrà messa in esecuzione;
- **Sicurezza**, visto che l'host su cui viene eseguito lo user agent, è che manderà quindi in esecuzione l'applet, non sa nulla di chi l'applet ha sviluppato e deve aver fiducia per accettare di mandarlo in esecuzione.

Java ci offre buone caratteristiche per soddisfare entrambi i requisiti.

Gli strumenti lato client devono offrire supporto per l'interazione con l'utente: devono quindi gestire gli **eventi**, quali click del mouse, bottoni, campi in cui l'utente può introdurre testo, etc.

Anche le azioni delle applet sono ristrette all'interno di una **sandbox**, ovvero un'area del browser allocata specificamente per l'applet. I **browser** hanno in genere un SecurityManager definito dal produttore. Le politiche di sicurezza previste nei browser prevedono che applet scaricate dalla rete:

- Non possono leggere o scrivere **file** sulla macchina ospite;
- Non possono creare **connessioni di rete** con host diversi dal server da cui sono stati scaricati;
- Non possono caricare **librerie** o essere composte da metodi nativi;
- Non possono avviare nuovi **processi** sulla macchina ospite;

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

- Non possono avere accesso a **informazioni** sulla macchina locale, tranne la versione di Java in uso, il nome e la versione del sistema operativo e alcune informazioni sui caratteri utilizzati; in particolare, non possono accedere al nome utente, al suo indirizzo di posta elettronica, etc,.
- Tutte le **finestre** aperte da un applet visualizzano un messaggio di avvertimento.

Un altro metodo per distribuire applets attraverso la rete è di usare **Java Web Start**. Programmazione lato client: tuttavia scaricare le applet è relativamente costoso, anche perché occorre una richiesta diversa per ogni classe. Per questo motivo risulta conveniente impacchettare tutti i file .class ed eventuali immagini e/o suoni in un unico file JAR (Java Archive) compresso che può essere scaricato con una sola richiesta.

Si noti che, se i browser sono abilitati per Java, le applets non danno problemi di installazione e che non c'è il rischio di provocare danni nella macchina del client a causa di qualche errore nel codice.

Essenziali per l'implementazioen di applet sofisticati sono la padronanza delle librerie di supporto alla rete e al multi-threading.

- Originariamente le applet venivano richiamate nel file HTML, attraverso l'etichetta <APPLET> con opportuni paramentri.
- Il W3C ne ha sconsigliato l'uso, preferendo l'etichetta <OBJECT> (HTML 4.0)
- I browser adesso supportano entrambe le etichette, ma le versioni più vecchie supportano solo <APPLET>.

In generale, applicazioni Web basate su applet Java sono molto usate nelle intranet aziendali (uso uniforme del browser per l'utente e facilità di gestione e aggiornamento per l'amministratore). Sulla intranet, l'uso del plug-in è perfettamente accettabile e permette di mantenere il controllo sulla piattaforma Java senza problemi di portabilità.

Un'applet è un'istanza della classe java.applet.Applet. Ecco un esempio di applet implementato così:

```
//very simple applet
//</applet>
import java.awt.*;
import javax.applet.*;

public class HelloWorldApplet extends Applet{
        public void paint(Graphics g) {
                g.drawString("Hello World!", 20, 20);
```

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

```
        }
    }
```

Se invece si usa Swing, l'applet si ottiene come istanza della classe JApplet, che è la superclasse delle applet Swing e una sottoclasse immediata della classe Applet.

```
//very simple applet, con Java 2 e le librerie Swing
//</apple>
import javax.swing.*;
import java.awt.*;

public class Applet1 extends JApplet {
    public void init() {
        getContentPane().add(new JLabel("Buongiorno!"));
    }
}
```

5.2.1. **Ciclo di vita di una applet**

Quattro metodi della classe Applet:

1. **init()**: viene chiamato automaticamente dal sistema quando l'applet comincia l'esecuzione; comprende quindi tutte le inizializzazioni;
2. **start()**: viene chiamato dopo init() e ogni volta che l'utente torna alla pagina contenente l'applet; quindi, mentre init() viene chiamato una sola volta, start() può venir chiamato più volte; di solito è da qui che viene riavviato il thread.
3. **stop():** non è necessario richiamarlo esplicitamente, in quanto viene richiamato ogni volta che l'utente esce dalla pagina in cui si trova l'applet; serve a sospendere tutte quelle attività che consumerebbero risorse inutilmente;
4. **destroy():** viene richiamato quando il browser viene chiuso normalmente, dopo aver chiamato stop(); non occorre distruggere il pannello.

Tra gli attributi dell'etichetta **<APPLET>** ricordiamo:

CODE: file.class corrispondente all'applet, a partire dalla pagina corrente; è obbligatorio indicare o CODE o OBJECT, che contiene l'oggetto applet serializzato;

CODEBASE: (facoltativo) indica la directory in cui si trovano i file relativi alle classi;

ARCHIVE: (facoltativo) il file o i file (separati da virgola) JAR, contenenti classi e altre risorse dell'applet; questi file vengono scaricati prima del caricamento dell'applet;

NAME: è possibile associare un nome all'applet in modo da poter chiamare alcuni dei suoi metodi da altri punti della pagina tramite del codice JavaScript o per far comunicare due applet.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Un'applet **NON** può comunicare con un'applet su una pagina diversa.

Il metodo getApplets() restituisce un **oggetto enumerazione**.

L'attributo **CODETYPE** specifica la natura dell'oggetto (application/java per le applet Java).

Naturalmente è possibile passare dei **parametri di ingresso** all'applet, usando l'etichetta HTML PARAM, con attributi definiti dal programmatore. I parametri sono sempre restituiti come stringhe, che devono poi essere convertite nel tipo richiesto. Il nome dell'attributo deve corrispondere esattamente al parametro passato al metodo getParameter, maiuscole e minuscole comprese. Se un parametro è stato omesso, getParameter restituisce **null**.

Può essere utile ricordare che se un'applet apre una finestra di pop-up, essa comprenderà sempre un **messaggio** che avverte che si tratta di una finestra aperta da un applet. Il messaggio può essere evitato solo se richiesto da una applet firmata. Ovviamente le applet possono elaborare sia **immagini** che **audio**, attraverso file che spesso sono identificati dai relativi URL.

5.2.2. **JNLP e Java Web Start**

Le applet firmate possono essere abilitate a essere eseguite con molte meno restrizioni delle applet qualsiasi e quindi possono in qualche modo prendere il posto delle normali applicazioni.

Esse hanno sempre il pesante vincolo di dover essere eseguite all'interno di un browser:

- con un maggior carico computazionale per il client che deve mandare in esecuzione anche il browser;
- con un impatto visivo non trascurabile, visto che vengono visualizzati anche tutti i pannelli di comandi del browser.

JNLP = Java Network Launch Protocol, permette di scaricare ed eseguire una applet anche fuori dal browser. Un'applicazione JNLP può scaricare dinamicamente risorse da Internet durante l'esecuzione.

Come per le applet, esse possono venir scaricate in file JAR firmati, in modo da dar modo all'utente di fidarsi di chi firma. Inoltre, anche quando vengono scaricate senza firma elettronica, possono chiedere il permesso di accedere ad alcune risorse del sistema del client attraverso le API JNLP.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

JNLP è un protocollo: ne serve un'implementazione, ad esempio **Java Web Start** (**JAWS**).

Un'applicazione basata su JNLP prevede un JAR file contenente un'applicazione standard, e un file XML per indicare al client come scaricare e installare l'applicazione.

6) Servlet

Un primo esempio di programmazione in Java lato server, strettamente collegato alle Java Server Pages o **JSP**. Trattandosi di programmazione Java, l'accesso alla base di dati, tipica delle applicazioni web standard, può venir implementata tramite Java DataBase Connectivity o **JDBC**.

6.1. **Introduzione**

Le servlet sono un esempio dell'uso di Java per la programmazione Web lato server. Rispetto al CGI presentano vantaggi in termini di efficienza, di facilità di sviluppo, di accesso all'informazione di contesto.

Occorre che il server sia predisposto all'esecuzione delle servlet, e preveda quello che si chiama **servlet engine** o **servlet container**.

6.1.1. **Confronto tra serlvet e CGI**

Possono essere viste come un secondo passo nell'andamento della programmazione lato server alle caratteristiche proprie delle applicazioni Web, in cui il server tipicamente riceve molte richieste in contemporanea e deve gestirle senza problemi di sincronizzazione e con un occhio alle prestazioni, sia in termini di tempo che di spazio.

Le servlet permettono di **condividere delle informazioni** sia all'interno di un'applicazione che col server. A differenza delle CGI, l'unico ambito di definizione (o scope) previsto è quello di una richiesta e per condividere informazioni tra due diverse richieste serve salvarle in modo persistente.

Un altro problema del CGI è che non da modo ai programmi CGI di interagire direttamente col server o comunque di usare le funzionalità del server dopo che il programma è entrato in esecuzione.

Nelle servlet, ad ogni richiesta corrisponde un nuovo thread all'interno del servlet engine: thread diversi possono accedere a variabili comuni, e questo permette di

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

condividere informazioni tra richieste diverse alla stessa servlet. Ovviamente ci possono essere problemi di sincronizzazione tra richieste/thread diversi.

Come per il CGI, una **servlet** permette di implementare delle funzionalità eseguite lato server. Quando un Web server riceve un richiesta per un programma CGI, esso:

- da origine ad un nuovo processo per eseguire il programma;
- passa a questo processo le informazioni necessarie per costruire la risposta via standard input e variabili d'ambiente.

La creazione di un nuovo processo per ogni richiesta ricevuta richiede risorse sia in termini di tempo che di memoria da parte del server, e limita quindi le risposte che possono essere elaborate contemporaneamente.

6.1.2. **Come si presenta una servlet**

Le **servlet** sono delle classi Java che possono venir caricate dinamicamente per estendere le funzionalità del server.

Le servlet vengono gestite all'interno della Java Virtual Machine sul server o operano solo sul server.

Le servlet vengono gestite all'interno del processo del server da thread separate. Questo presenta due vantaggi:

- efficienza e scalabilità;
- ottima interazione col server.

Le **Java Server Pages** (JSP) vengono tradotte in servlets e forniscono quindi un sistema per implementare il meccanismo di richiesta e risposa del server senza doversi occupare di tutti i dettagli implementativi delle servlets.

Normalmente, si usano:

- le **JSP** quando la maggior parte del contenuto passato al client è testo statico o markup;
- le **servlets** quando la porzione di testo da produrre staticamente è poca.

Le servlet normalmente passano il loro risultato al client sotto forma di file HTML, XHTML, XML da visualizzare nel browser , ma anche in altri formati, quali immagini e dati binari.

Tutte le servlet devono implementare l'interfaccia Servlet, che è la classica API di un fornitore di servizi.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Ogni classe può implementare un numero qualsiasi di interfacce. Per implementare un'interfaccia occorre che la classe definisca un'implementazione per tutti i metodi dell'interfaccia, in modo che i prototipi coincidano.

I metodi dell'interfaccia Servlet vengono invocati automaticamente dal servlet engine.

```
    public void init(ServletConfig config)
                              throws ServletException;
    public ServletConfig getServletConfig();
    public void service(ServletRequest req, ServletResponse res)
                             throws ServletException, IOException;
    public String getServletInfo();
    public void destroy();
}
```

L'interfaccia dichiara quindi cinque metodi, di cui però non dà il body e quindi l'implementazione:

1. **void init(ServletConfig config)**: inizializza la servlet una volta sola durante il ciclo di esecuzione della servlet;
2. **ServletConfig getServletConfig()**: ritorna un oggetto che implementa l'interfaccia ServletConfig e fornisce l'accesso alle informazioni sulla configurazione della servlet, tra cui i paramentri di inizializzazione e il **ServeltContext**, ovvero il suo ambiente;
3. **String getServletInfo()**: viene definito in modo da restituire una stringa con informazioni quali autore e versione;
4. **void service(ServletRequest request, ServletResponse response)**: viene mandato in esecuzione in risposta alla richiesta mandata da un client alla servlet;
5. **void destroy()**: invocato quando la servlet viene terminata; usato per rilasciare le risorse (connessioni a basi di dati, file aperti, etc.)

Il metodo **init()** viene invocato solo quando la servlet viene caricata in memoria, di solito a seguito della prima richiesta per quella servlet. Solo dopo che il metodo è stato eseguito, il server può soddisfare la richiesta del client invocando il metodo **service()**, che riceve la richiesta, la elabora e prepara la risposta per il client. Quindi il metodo **service()** viene richiamato ogni volta che si riceve una richiesta. Tipicamente, per ogni nuova richiesta il servlet engine crea una nuova thread di esecuzione in cui viene mandato in esecuzione **service()**. Solo quando il servlet engine termina la servlet, viene invocato il metodo **destroy()** per rilasciare le risorse della servlet.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Ecco il guadagno in efficienza rispetto al CGI: una thread invece di un nuovo processo per ogni richiesta.

I pacchetti delle servlet definiscono due classi astratte che implementano l'interfaccia Servlet:

- **GenericServlet** (in javax.servelt);
- **HttpServlet** (in javax.servlet.http).

Queste classi forniscono un'implementazione di default per tutti i metodi dell'interfaccia.

```
public abstract class {\bf HttpServlet}
          extends GenericServlet implements java.io.Serializable
```

Questa funzione è molto interessante, implementa una servlet rispetto al protocollo HTTP, e quindi per il web.

Ogni sottoclasse di **HttpServlet** deve fare overriding di almeno un metodo, di solito uno dei seguenti:

- doGet, se la servlet viene richiamata col metodo HTTP GET;
- doPost, per richieste HTTP con metodo POST;
- doPut, per richieste HTTP con metodo PUT;
- doDelete, per richieste HTTP con metodo DELETE;
- init e destroy, per gestire eventuali risorse necessarie al funzionamento della servlet (ad esempio, connessioni con la base di dati);
- getServletInfo, usata dalla servlet per fornire informazioni riguardanti se stessa.

Una strategia tipicamente utilizzata è di non riscrivere il metodo **service()**, ma piuttosto i metodi "doXXX".

Il metodo più significativo in una servlet è ovviamente il metodo **service()**, o in alternativa di metodi **doXXX**, che prendono in ingresso:

- un oggetto **ServletRequest**, corrispondente ad uno stream di ingresso, da cui leggere la richiesta del client;
- un oggetto **ServletResponse**, corrispondente ad uno stream di uscita, su cui scrivere la risposta.

In caso di problemi, viene lanciata una **ServletException** o una **IOException**. In caso di richieste multiple, diverse chiamate al metodo **service()** verranno eseguite in **parallelo**.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Il funzionamento tipico delle servlet è basato su multithreading: ne consegue la necessità di gestire richieste concorrenti e la sincronizzazione nell'accesso alle risorse.

6.1.3. **La classe HttpServlet**

Le servlet per il Web tipicamente estendono la classe **HttpServlet**.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class ServletsRule extends HttpServlet {
   int i = 0; // Servlet "persistence"
   public void service(HttpServletRequest req, HttpServletResponse res) \
                                                        throws IOException {
      res.setContentType("text/html");
                  // {\bf prima} di inizializzare un Writer o un OutputStream
      PrintWriter out = res.getWriter();
             // in alternativa, getOutputStream( ) produce un OutputStream, usato
                                    // per risposte binarie
      out.print("<HEAD><TITLE>");
      out.print("A server-side strategy");
      out.print("</TITLE></HEAD><BODY>");
      out.print("\section{Servlets Rule! " + i++);
      out.print("}</BODY>");
      out.close();
   }
}
```

Per non perdere l'informazione quando il servlet engine viene, per qualsiasi motivo, terminato, si utilizzano i metodi **init()** e **destroy()**, che vengono chiamati automaticamente quando la servlet viene caricata e scaricata, e in ogni caso in cui il servlet engine viene terminato.

Il metodo **service()** viene sovrascritto per tener conto delle diverse richieste che arrivano ad un client Web. Il metodo **service()** prima di tutto decide quale richiesta è stata inoltrata, e poi chiama il metodo opportuno: ad esempio, **doGet()** o **doPost()** a seconda del tipo di richiesta.

Altre chiamate di uso molto meno frequente sono: **doDelete**, **doOptions**, **doPut**, **doTrace**.

Tutti questi metodi ricevono in ingresso due parametri:

- **HttpServletRequest request**: che rende accessibili i dati relativi alla richiesta inoltrata dal client;
- **HttpServletResponse response**: per la risposta da passare al client.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Hanno un tipo di ritorno void.

7) CMS e Web Frameworks

Le moderne applicazioni per il web hanno molte caratteristiche in comune con le classiche applicazioni desktop, sia in termini tecnologici, che in termini di esperienza e interazione utente, tanto da guadagnarsi la denominazione di Rich Internet Applications (**RIA**).

Le RIA utilizzano tecnologie lato client in maniera intensiva, quali Javascript e CSS3 in combinazione con AJAX per l'invio asincrono di richieste HTTP, per la realizzazione di interfacce utente sempre più simili a quelle desktop.

Lo sviluppo e la diffusione delle RIA hanno sancito la nascita del cosiddetto Web 2.0, acquisendo la denominazione di soluzioni batterie incluse, vale a dire solo soluzioni integrate.

In questo contesto vanno considerati i Content Management Systems (**CMS**) e i Web Application Frameworks:

- I **CMS** hanno lo scopo di gestire workflows e lo sviluppo di contenuti in ambienti di lavoro collaborativo e vengono quindi progettati per semplificare la pubblicazione di documenti su web.
- I **Web Frameworks**, invece si prefiggono lo scopo di alleggerire il lavoro legato allo sviluppo di applicazioni web, fornendo supporto per le funzionalità tipicamente richieste per la loro realizzazione quali la gestione e la visualizzazione dei dati.

Con riferimento alle RIA, infine è possibile enfatizzare la differenza che sussiste tra CMS e Web Frameworks: i CMS **sono RIA**, mentre i Web Frameworks mettono a disposizione gli **strumenti** per lo sviluppo delle RIA

7.1 **Content Management Systems, CMS**

Un CMS è un sistema per la gestione dei contenuti. Si propone di organizzare il lavoro sui contenuti di un gran numero di utenti. Ovviamente agli utenti non viene richiesta nessuna competenza informatica o web, al di là di saper utilizzare un generico browser.

La maggior parte dei CMS usa una base di dati relazionale per immagazzinare sia i contenuti che i metadati che qualsiasi altro contenuto necessario al sistema.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

I contenuti sono spesso immagazzinati come XML e oltre a documenti testuali, può gestire anche documenti multimediali: immagini, file audio, video, etc. e anche files di dati.

Un CMS offre supporto alla creazione, alla modifica e alla pubblicazione dei documenti.

È generalmente possibile anche sviluppare un documento in collaborazione tra più persone, con ovvi problemi di sincronizzazione e di controllo delle versioni. Si dà la possibilità anche di importare documenti legacy o eventualmente esterni nel sistema, eventualmente anche mediante l'uso di **OCR**. È poi previsto anche un sistema di controllo delle versioni.

Un'altra importante funzionalità offerta dai CMS è il supporto al **retrieval** dei documenti, con metodi di ricerca che possono variare da sistema a sistema: ricerca per parole chiave, query booleane.

I documenti immessi nel CMS devono quindi venir organizzati in una repository con indicizzazioni opportune, in modo da facilitare l'accesso.

Alcuni CMS insistono anche sulla possibilità di sperare i contenuti dalla formattazione, permettendo la configurazione degli aspetti grafici.

Controllo degli accessi: un aspetto importante che un CMS deve essere in grado di gestire, definendo diverse categorie di utenti con diversi profili e quindi permessi di accesso.

Workflow: supporto al lavoro in collaborazione che segue il percorso di un documento elettronico prevedendo una serie di azioni ad esso associate da parte di diverse persone.

One-to-one marketing: incluso da molti dei CMS, prevede strumenti per personalizzare la pagina per lo specifico utente.

7.2. **Cosa si intende per Web Framework**

Il termine **framework** è uno dei termini più usati e abusati in informatica, a cui si attribuiscono spesso differenti significati e accezioni. Esistono diverse tipologie di framework: framework di **testing**, di **sviluppo**, **web** framework, ecc.

Un **framework** non è né un'API, che specifica come le componenti software devono interagire l'uno con l'altra, né una libreria, vale a dire una raccolta di funzioni o oggetti destinati ad un particolare scopo. Un framework non è un'applicazione, ma

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

uno **strumento che offre supporto allo sviluppo di applicazioni**, mettendo a disposizione una serie di strumenti e soluzioni integrate già pronte all'uso.

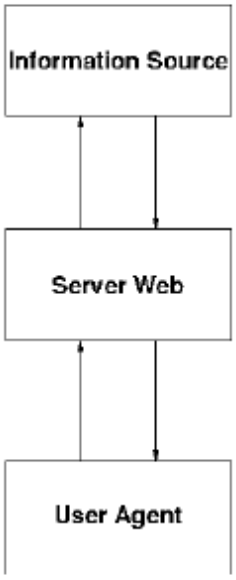
Un **web framework**, dunque, non è altro che un **framework per lo sviluppo di applicazioni specifiche per il web**, mettendo a disposizione sia strumenti per la programmazione lato server, che per la programmazione lato client.

Nascono dalla consapevolezza che le operazioni svolte da uno sviluppatore con la programmazione lato server sono tipicamente sempre le stesse.

7.2.1. Architettura delle applicazioni web

Un'**architettura** (software) descrive la **struttura** di un'applicazione, la decomposizione in **componenti**, le loro **interfacce** e **interazioni.** L'architettura rappresenta il passaggio tra analisi e implementazione. Le applicazioni Web sono un esempio di architettura **multi-tier** (multistrato) o **n-tier**. Per **tier** si intende un raggruppamento logico di funzionalità. I diversi tier possono stare sulla stessa macchina o su macchine diverse.

Le applicazioni Web più semplici sono a tre tier, quindi **three-tier**



Il tier intermedio controlla l'interazione tra client e sorgente di informazione; implementa:

- Il meccanismo di controllo (**controller logic**): elabora la richiesta del client e si procura i dati da presentargli;
- Il meccanismo di presentazione (**presentation logic**) elabora questi dati per presentarli all'utente (in HTML, XHTML, WML,…);

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

- Le regole del dominio applicativo (**business logic**) e si assicura che i dati siano affidabili prima di usarli per aggiornare la sorgente di informazione o per rispondere all'utente; le **business rules** regolano l'accesso dei client ai dati e l'elaborazione dei dati da parte dell'applicazione.

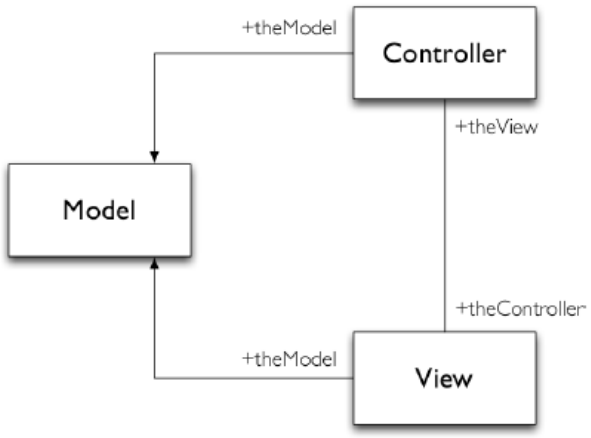
7.2.2. **Modello MVC e Web Frameworks**

Uno degli aspetti maggiormente rilevanti all'impiego di un framework per lo sviluppo delle applicazioni (web) riguarda il dover adattare il proprio modello di design a quello "imposto" dal framework utilizzato. Per ciò che riguarda nello specifico i web frameworks, potrebbe essere quella di adottare la solita architettura software a tre strati.

Questa struttura è caratterizzata da una forte dipendenza e da un alto grado di accoppiamento tra i vari strati e risulta quindi non sufficientemente modulare da poter essere sistematizzata in un framework.

Il requisito di modularità risulta essere di fondamentale importanza per la realizzazione di un (web) framework. Il modello architetturale a 3-tier, infatti, presenta un'integrazione troppo stretta tra livello di presentazione (View o Vista), di elaborazione (Controller o Controllore) e l'acquisizione dei dati dalla sorgente di informazione, che tipicamente è una base di dati (Model o Modello).

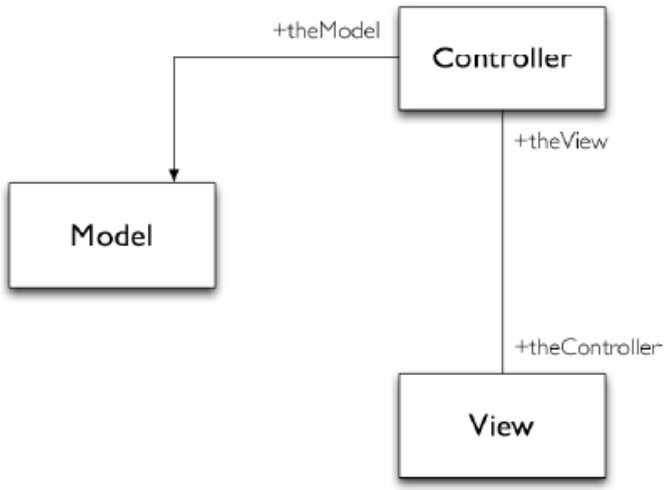
Ciascuno dei tre strati richiede competenze diverse, questo rende più difficile la spartizione delle responsabilità all'interno di un gruppo di lavoro.



Il modello **MVC**, ModelViewController, è stato adattato dal pattern precedente.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Nella versione Web, non vi è alcuna relazione diretta tra modello e vista, a causa del fatto che la comunicazione tra essi è veicolata da richieste HTTP che è notoriamente un protocollo **stateless** (senza stato). Il pattern MVC per il web è a volte indicato con il termine **WebMVC**



In questa versione del pattern, il modello ha la responsabilità della gestione dell'acquisizione dei dati e del dominiio, il controllore dell'elaborazione dei dati e della comunicazione tra le diverse componenti dell'applicazione, mentre la vista è la sola responsabile della verbalizzazione e presentazione dei dati.

I tre diversi livelli di astrazione del pattern (Web-)MVC sono delegati ad assolvere specifiche funzionalità.

Model: centralizza il controllo e l'accesso ai dati, rendendo fruibile al livello di controllo la gestione dei dati. Usualmente lo scambio dei messaggi tra il livello "modello" e "controllo" è veicolata da un **ORM** (Object Relational Mapping) responsabile del mapping tra modello dei dati e corrispondente modello ad oggetti dell'applicazione. Si fa carico quindi dell'interfacciamento con la basi di dati.

View: è responsabile della composizione e visualizzazione delle risorse, spesso rappresentate da pagine HTML, ma anche XML. Questo livello integra tipicamente librerie JavaScript per la programmazione lato-client e meccanismi di composizione e aggregazione delle pagine web, attraverso sistemi di template per la compilazione dinamica delle risorse web.

Controller: ha la responsabilità di acquisire le richieste HTTP, gestire la comunicazione con il modello dei dati e di traferire i dati alla "vista" per la compilazione delle risorse. Caratteristiche tecniche relative a tale livello comprendono:

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

1. **URL Routing RESTful**: meccanismo di indirizzamento degli URL supportati dalla applicazione web, che rispetti le linee guida della specifica REST (**RE**presentational **S**tate **T**ransfer);
2. **Meccanismi di sicurezza** contro attacchi quali SQL-Injections, Clickjacking e CSRF.

I web framework, forniscono supporto per tutti i livelli del pattern Web-MVC, mettendo a disposizione dello sviluppatore uno stack di componenti software. Difatti, si parla di full-stack frameworks per riferirsi a quei framekworks che offrono supporto, attraverso componenti propriamente implementate o integrate da strumenti di terze parti, per ciascun livello dello stack software.

7.2.3. **Micro-stack framework**

Non offrono alcun supporto out-of-the-box per i livelli di modello e vista, ma offrono solo l'infrastruttura in grado di gestire e smistare le richieste HTTP per l'applicazione web (controller).

Tuttavia, entrambe le tipologie di web framework, aggiungono allo stack software un web server di sviluppo integrato, da impiegare esclusivamente e tassativamente solo per lo sviluppo, ma è progettato per sollevare lo sviluppatore dal dover necessariamente configurare l'infrastruttura in grado di elaborare le richieste HTTP.

7.3. **Principi di progettazione e scopi**

Tutti i web framework, a prescindere dalla tipologia (full o micro stack), condividono alcuni principi di design e finalità.

- **Convention over Configuration:** principio che favorisce l'adozione di un insieme di "convenzioni" (di progettazione e implementazione) piuttosto che la configurazione nel dettaglio dell'applicazione; ad una classe di modello denominata ExampleModel dovrà necessariamente corrispondere un relativo ExampleModelController.
- **DRY (Don't Repeat Yourself):** si tratta di un principoi generale che sconsiglia la duplicazione dell'informazione e insiste nel mantenere ogni parte dell'informazione in un unico punto del sistema.
- **Accoppiamento lasco:** i diversi livelli del framework non dovrebbero avere conoscenza approfondita l'uno dell'altro se non nei casi in cui questo sia assolutamente necessario.
- **Minor quantità di codice:** non dovrebbe essere necessario sviluppare codice per le parti scontate e ripetitive. Lo sviluppo delle applicazioni deve richiedere

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

meno codice possibile, dato che la maggior parte del codice è a carico del framework.

- **Sviluppo veloce:** in un certo senso questo è il punto principale che spiega l'esistenza dei web frameworks, ovvero di velocizzare lo sviluppo su web, in particolare evitando o riducendo il più possibile gli aspetti più noiosi e ripetitivi, assicurando meccanismi di gestione già sistematizzati e testati.

8) WebUML

8.1. **Web engineering**

La progettazione di sistemi basati su web ha una serie di pecurialità da un punto di vista dell'ingegneria del software.

- I requisiti sono altamente instabili;
- È richiesta molta più interazione con i committenti;
- Lo sviluppo è caratterizzato da forti pressioni sui tempi, con conseguente compressione dello scheduling;
- Le tecnologie cambiano con frequenza molto maggiore che in ambito desktop;
- Non si riesce a sfruttare pienamente il paradigma orientato agli oggetti;
- Sviluppati da team altamente eterogenei, con differenti competenze;
- Non esistono strumenti di sviluppo solidi come quelli in ambito desktop.

8.2. **Web engineering: estensione di UML per il Web di Conallen**

I sistemi basati su web non sono altro che un tipo particolare di sistemi client-server, in cui gli elementi principali sono client, server e rete di connessione.

La maggior parte dei sistemi basati su web considerano utenti anonimi, che non hanno alcuna abilità o competenza informatica.

Nella progettazione sono quindi molto importi i casi d'uso (**use cases**).

Il linguaggio di modellazione UML è stato ideato per l'analisi e la progettazione di sistemi orientati agli oggetti e permette di rappresentare graficamente i sistemi o mediante diagrammi strutturali, basati su classi, componenti, oggetti, o mediante diagrammi comportamentali per i casi d'uso, sequenze, collaborazioni, statechart e attività.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

8.2.1. **Peculiarità dei sistemi basati su web**

Un aspetto importante di UML è che prevede dei meccanismi per estenderlo.

Ovviamente questa struttura va usata con attenzione: estensioni introdotte con superficialità possono facilmente condurre a problemi di inconsistenza dei modelli. Sembra opportuno mantenere la consistenza anche con la semantica del cuore del linguaggio e di possibili altre estensioni.

Scegliere un livello di astrazione troppo alto o troppo basso conduce a modelli che risultano fuorvianti o perché omettono importanti dettagli o perché ne riportano troppi, facendo perdere di vista la struttura fondamentale del progetto.

Gli schemi architetturali di base per le applicazioni web usano le pagine web come elemento di base.

Le pagine client sono artefatti che possono essere considerati come ogni altra interfaccia utente di un sistema.

Le pagine server interagiscono con le risorse lato server prima di essere inviate al client come interfaccia utente completa.

Non essendo un approccio specificamente orientato agli oggetti, UML non riesce a modellare bene questo importante aspetto dei sistemi web.

La notazione di UML così come vi è stata presentata non è sufficiente a descrivere le pagine server come oggetti nel class diagram. L'unica soluzione è quindi quella di usare gli strumenti previsti da UML e costruire un'estensione che permetta di includere questi aspetti di sistemi web.

Le estensioni sono rese possibili attraverso un meccanismo che mette assieme:

- **Stereotipi;**
- **Tagged values;**
- **Vincoli o constraints impliciti** che modificano la semantica.

Quello degli stereotipi è un meccanismo di estensioni che consente di classificare i model element di UML. Uno stereotipo S si definisce con **<< S >>**

I tagged values rappresentano un'estensione delle proprietà associate ad un elemento del modello. È costituita da una coppia che permette di aggiungere informazioni arbitrarie ad ogni elemento del modello. (**Tag, Value**).

I vincoli rappresentano un'estensione della semantica del linguaggio. Si tratta di un ruolo attribuito ad un elemento del modello che restringe la sua semantica. Consentono di rappresentare fenomeni che altri non potrebbero essere espressi con

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

UML. Un vincolo è un modo per definire come i modelli possono essere messi insieme.

8.2.2. **L'estensione di Conallen**

Usare UML per modellare un sistema web richiede di introdurre il concetto di pagina. Per risolvere questo punto, l'estensione di UML proposta da Conallen propone di introdurre due **stereotipi**: **la pagina server** e **la pagina client.**

La prima contiene metodi e variabili relativi allo **scripting server-side** (CGI, JSP, ecc.), la seconda contiene elementi relativi alla formattazione, allo **scripting client-side** (Applet, ActiveX, JavaScript, ecc.)

Ai modelli usualmente usati dagli altri tipi di sistemi software introduciamo anche le **site map**, un'astrazione dell'insieme delle pagine Web e dei collegamenti di navigazione all'interno del sistema.

Occorre modellare le **pagine**, i collegamenti (**link**) tra di esse, il **contenuto dinamico** necessario alla loro costruzione e il contenuto dinamico che arriva fino al client.

WebUML si occupa di modellare la **business logic** e non la **presentation logic**.

UML si basa sulla modellizzazione in classi, e quindi non può essere utilizzato così com'è: una pagina Web assomiglia più ad un componente che ad una vera e propria classe.

Occorre ora far corrispondere ciascuno di questi quattro elementi dell'applicazione ad un diverso elemento del modello.

Hyperlinks: associazione tra elementi.

Pagine: classi (con tre aree: nome, attributi e operazioni).

Scripts: operazione nella classe.

Variabili: definite con scope di pagina; attributi della classe.

UML può venir esteso tramite l'uso di **stereotipi** che ci permettono di associare un nuovo significato ad un elemento del modello, **tagged values**, ovvero coppie chiave-valore da associare ad elementi del modello e **contraints** che permettono di definire delle regole necessarie a che il modello sia ben formato.

Nell'estensione di UML di Conallen, **ogni pagina Web**, sia essa statica o dinamica, **viene modellata come un componente**. La Implementation View o Component View descrive i componenti del sistema e le relazioni che intercorrono tra loro.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Corrispondono quindi alle pagine Web e ai relativi collegamenti dando origine a una mappa del sito (**site map**).

Il comportamento di una pagina Web sul server è completamente diverso che sul client: ciascuno di questi due aspetti di ogni pagina verrà quindi modellato separatamente, e la relazione tra le due viene stereotipata come **build**, nel senso che la **pagina lato server "costruisce" quella lato client**.

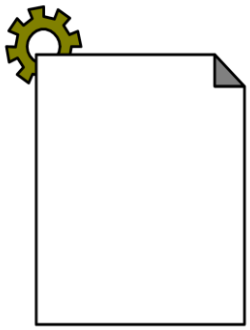
Ovviamente mentre ogni pagina lato client può venir costruita da al più una pagina lato server, una pagina lato server può costruire anche più pagine lato client.

In un'applicazione Web, un link permette di navigare da una pagina all'altra. Nel modello questa relazione viene modellata da una associazione stereotipata da tipo **link**. Un'associazione di questo tipo ha sempre origine da una pagina lato client e punta ad un'altra pagina che può essere sia lato client (ad esempio, pagina statica) che lato server (ad esempio, una JSP, una servlet o una CGI).

I tagged values permettono di associare dei parametri ai link.

Server Page

Si tratta di una pagina web contenente script eseguiti dal server, che interagiscono con risorse lato server come database, business logic, sistemi esterni. Le operazioni della classe rappresentano le funzioni dello script, gli attributi rappresentano le variabili visibili nello scope della pagina.

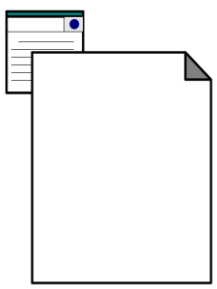


Unico vincolo di avere relazioni soltanto con oggetti sul server.

I tagged values di una pagina server descrivono lo scripting engine, ovvero il linguaggio o il motore che dovrebbe essere utilizzato per eseguire o interpretare questa pagina.

Client page

Si tratta di una pagina direttamente rappresentabile dallo user agent, quale ad esempio il browser. Può contenere script che sono interpretati dallo user agent. Le operazioni corrispondono alle funzioni degli script, mentre gli attributi corrispondono alle variabili.



Una client page può avere associazioni con altre pagine, client o server.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Non è soggetta a nessun vincolo e ammette i seguenti tagged values:

- **Titolo-tag:** il titolo della pagina è mostrato dal browser;
- **Base-tag:** URL di base per dereferenziare relativi URLS;
- **Body-tag:** l'insieme degli attributi per l'elemento <body> che impostano il background e gli attributi di testo di default.

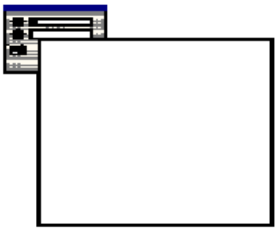
Per esprimere un collegameto tra una pagina client ed una server si usano i links. Nel caso opposto abbiamo i redirect, oppure build.

Form

Le form rappresentano una componente caratteristica di moltissime pagine client e quindi bisogna chiedersi come rappresentarle. Contengono attributi addizionali rispetto alle normali pagine. È possibile che una pagina contenga più form. Un form può essere rappresentato come una classe i cui attributi sono i campi del form.

La relazione tra pagina e form è un'aggregazione.

Lo stereotipo della realzione tra in forme la pagina server che ne utilizza i dati è un submit.



Si tratta di una serie di campi di input che sono parte di una pagina client. Corrisponde all'etichetta HTML <form>. Gli attributi di questa classe sono rappresentati dai campi di input (input box, text area, radio buttons, check boxes, campi hidden) dell'elemento.

Un form non possiede operazioni né vincoli.

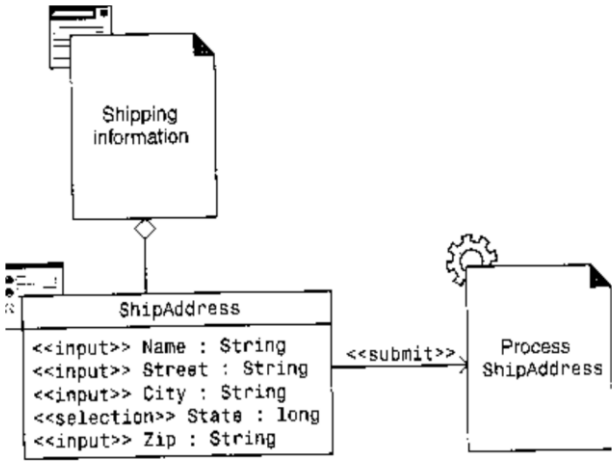
Il metodo (ad esempio, GET o POST) usato per sottomettere i dati nella action URL viene rappresentato come tagged value.

Submit

Si tratta di un'associazione che associa un form ad una server page, alla quale il form invia i dati. Sono elaborate le pagine server che utilizzano le informazioni nel form sottoposto.

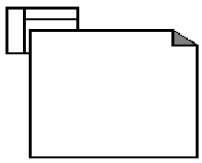
Non ha nessun vincolo, ma un tagged value per i parametri: una lista di nomi di parametri che dovrebbero essere passati con la richiesta della pagina destinazione dell'associazione.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it



Frameset

Viene rappresentata come una classe contenitore di pagine web multiple. L'area viene divisa in frame, ed ogni frame viene associato ad al più un << target >> (anche nessuno). Il contenuto di ciascun frame può essere una web page o un altro frameset.



Nessun vincolo.

Tagged value:

- **Rows:** il valore dell'attributo della riga del tag << frameset >>;
- **Cols:** il valore dell'attributo della colonna del tag << frameset >>.

Partizionamento lato client e lato server

Questa attività richiede di individuare le server page e le client page e di stabilire le relazioni tra loro e con gli altri oggetti del sistema. Le due attività principali nel design di applicazioni Web sono significativamente differenti dal design di altri sistemi software: Partizionamento degli oggetti lato client e lato server e definizione delle pagine web come interfacce utenti.

Per le applicazioni che usano un'architettura **Thin web client** tutti gli oggetti sono lato server (running on the web server or another tier associated with the server). Per le applicazioni che usano un'architettura **Fat web client**, in gran parte, oggetti persistenti, oggetti contenitori, oggetti condivisi, e oggetti complessi appartengono al lato server.

Oggetti candidati per il lato client sono gli oggetti che non hanno associazioni con risorse o oggetti lato server, ovvero che hanno dipendenze solo con risorse lato client.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

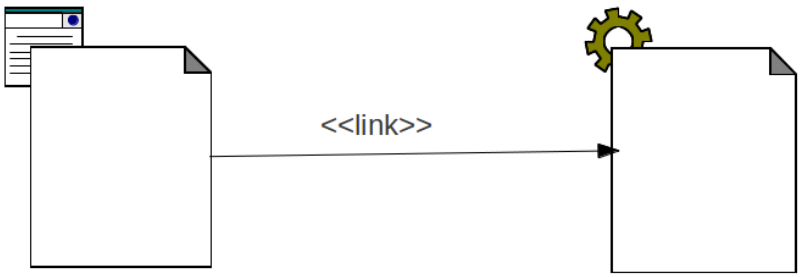
Inoltre, oggetti che contengono campi da validare, controlli per le interfacce utenti, controlli per la navigazione, etc.

Link

Tale stereotipo è definito per un'associazione tra pagine client e altre pagine (server o client): si noti però che anche se il link collega due pagine client, il server viene sempre coinvolto perché il link presuppone una richiesta http al server.

Un'associazione di tipo link può essere sia unidirezionale che bidirezionale, nel caso ci siano due link che collegano le due pagine nelle due direzioni.

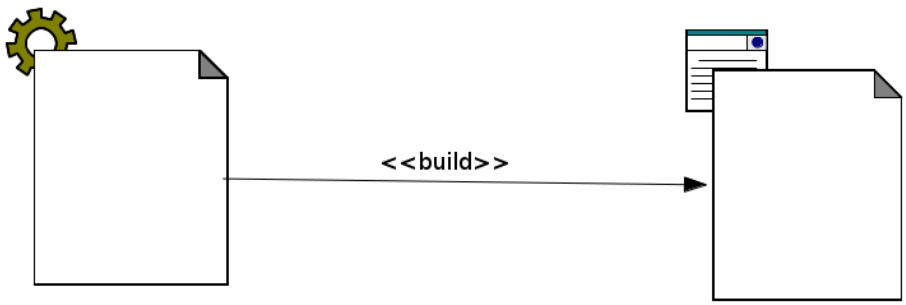
Corrisponde ad un'associazione UML. Rappresenta un puntatore tra un client page un'altra pagina. Ha un unico tagged value per i parametri: una lista di nomi di parametri che dovrebbero essere passati insieme al momento della richiesta per la pagina destinazione del link.



Build

Un'altra importante associazione è quella che collega una pagina server con la pagina client che essa produce in uscita (**build**). La relazione, indicata dallo stereotipo builds è unidirezionale.

Si tratta di una relazione unidirezionale, visto che la client page non contiene informazioni relative al perché è stata creata; inoltre, una server page può creare più client page, che invece sono create da una sola server page.



Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Redirect

Un'altra importante associazione è rappresentata dalla redirect, che indica il trasferimento del controllo da una pagina ad un'altra. In questo caso sia la sorgente che la destinazione possono essere client pages o server pages. Se l'origine è una client page, la page destinazione verrà automaticamente richiesta dal browser, allo scadere di un determinato intervallo di tempo.

Non ha nessun vincolo.

Ha come tagged value il delay che dà l'ammontare di tempo che la client page dovrebbe aspettare prima del redirect alla prossima pagina.

9) JSP

9.1. **JavaServer Pages (JSP)**

Le JSP (JavaServer Pages), sono una tecnologia Java che permettono allo sviluppatore di software, di creare pagine web dinamiche basate su HTML, XML o altri tipi di documenti.

La compilazione delle JSP avviene in modo automatico la prima volta che arriva una richiesta per la pagina e ogni volta che la pagina viene modificata, ma non occorre ricompilare se le modifiche riguardano solo il template statico.

Una pagina JSP è una pagina di testo formata da:

- Un template statico, espresso in un qualsiasi formato testuale;
- Istruzioni java.

La prima volta che una pagina JSP viene richiamata, essa viene tradotta in una servlet e viene messa in esecuzione dal servlet engine.

Viene prodotta una pagina Web da mandare al browser per essere visualizzata. La servlet può creare oggetti necessari alla comunicazione e scrivere le corrispondenti stringhe su uno stream di uscita che passa alla risposta HTTP.

Si possono avere due tipi di errori:

- Errori durante la **traduzione** (translation time);
- Errori durante l'elaborazione della **richiesta** (request time).

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Una pagina JSP ha in più alcuni elementi:

- **Direttive:** <%@ directive %>
  <%@ page import="java.util.*" %>;
  Consentono di influenzare la struttura e l'organizzazione della pagina JSP e di includere contenuti da altre risorse (è possibile caricare altre classi Java tramite l'attributo import per esempio). Vengono elaborate durante la traduzione in Servlet della JSP, pertanto sono indipendenti dalla singola richiesta.

- **Dichiarazioni:** <%! declaration %>
  <%! Private int accessCount=0; %>;
  Queste dichiarazioni all'interno dei tag sono globali! Nel senso che vengono aggiunte esternamente ad ogni metodo della Servlet generata a partire dalla JSP. Usate per lo più per la definizione di metodi e variabili di classe;

- **Scriptlet:** <% scriptlet %>
  Le scriptlet sono frammenti di codice che dovranno essere eseguiti quando sarà necessario processare una richiesta, la loro sintassi è così definita:
  <% codice %>
  Tutto il codice tra <% e %> verrà inserito nel metodo _jspService() della Servlet durante la compilazione, questo vuol dire che, le dichiarazioni fatte con <\%! sono condivise da **tutte le richieste**, mentre le dichiarazioni effettuate nelle scriptlet sono le dichiarazioni che sono presenti **solo nella richiesta** e non sono condivise.

- **Espressioni:** <%= expression %>
  <%= request.getRemoteHost()%>
  L'espressione racchiusa tra i tag viene valutata al momento della richiesta e il suo valore inserito direttamente nell'output generato dalla pagina JSP;

- **Librerie di etichette**

Il resto della pagina è detto di solito fixed-template data o fixed-template text.

All'interno di una pagina Web, posso introdurre la linea:

<\%= new java.util.Date() \%>

per inserire informazioni su data e ora. Quando il client richiede la pagina .jsp contenente queste espressioni, il server manda in esecuzione una servlet che crea

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

un oggetto di tipo Date dal pacchetto java.util e stampa la stringa corrispondente nella risposta da passare al client.

9.1.2. **Elementi di scripting**

Gli **elementi di scripting** possono essere usati non solo per produrre una pagina dinamica, ma anche per produrre pagine statiche da restituire solo se sono soddisfatti determinati requisiti.

Sono componenti di scripting:

- **Scriptlets:** <% … %>; le istruzioni contenute in uno scriptlet vengono messe in esecuzione durante l'elaborazione della richiesta HTTP;
- **Commenti:**
  - Commenti **JSP**: <%- … -%>, ovunque, ma non all'interno di uno scriptlet;
  - Commenti **XHTML**: <!- … ->, idem;
  - Commenti del **linguaggio di scripting**: in questo momento solo Java.
- **Espressioni:** <%= … =%>, contenente un'espressione Java, che il contenitore converte sempre in un oggetto String
- **Dichiarazioni:** <%! … !%>, per la defnizione di variabili e metodi, che diventano membri della classe ottenuta dalla traduzione della JSP;
- **Sequenza di escape:** <%, %>, ', '', \

9.1.3. **Oggetti impliciti**

Gli oggetti impliciti in JSP del container JSP sono previsti per ogni pagina di oggetti Java, gli sviluppatori possono usarli direttamente, senza una dichiarazione esplicita. Gli oggetti impliciti JSP sono anche noti come variabili predefinite.

I possibili ambiti definizione possono essere:

- **Scope di applicazione:**
  - application: il contenitore all'interno del quale viene eseguita la servlet;
- **Scope di pagina:**
  - config;
  - exception: solo in una pagina di errore;
  - out: consente di generare output dinamico bufferizzato da inviare al client;
  - page;
  - pageContext;
  - response: permette la manipolazione della risposta da inviare al client, per esempio reindirizzare il client ad uno specifico URL;

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

- **Scope di richiesta**
  - request: consente di accedere alle informazioni contenute nella richiesta HTTP inviata dal cliente come ad esempio parametri dei form e cookies;
- **Scope di sessione**
  - session: una delle più utilizzate, consente di accedere e manipolare la sessione degli utenti dell'applicazione.

Uno scriptlet può contenere qualsiasi numero di statement, variabili o dichiarazioni di metodi del linguaggio JAVA, o espressioni che sono valide nella pagina del linguaggio scripting.

Invece per gli elementi "dichiarazione" si possono dichiarare una o più variabili, o metodi che possono essere usati dopo nel file JSP. Bisogna quindi dichiarare la variabile o il metodo prima di ursarla/o nel file JSP.

9.3. **JavaBeans**

La tecnologia JSP permette inoltre immediato supporto alla programmazione per componenti JavaBeans.

JavaBeans sono componenti software riusabili per Java. Tecnicamente sono delle classi, scritte in Java, conformi a particolari convenzioni. Sono usati per incapsulare più oggetti in un unico oggetto (il bean) in modo che possano semplificare il passaggio di quest'ultimi.

Un JavaBeans è un oggetto Java che è serializable (in modo da poter salvare e ripristinare il suo stato in modo persistente), ha un costruttore con zero argomenti e consente accesso alle sue proprietà tramite metodi **SET** e **GET.**

Una JSP può creare e usare qualsiasi oggetto Java con un tag di dichiarazione o scriptlet. Se però l'oggetto in questione è conforme alle convenzioni JavaBeans la pagina JSP può usare i tag di azione standard per creare e accedere all'oggetto. Esempio: **<jsp:useBean id="cart" class="cart.ShoppingCart" scope="sessione"/>**

La differenza sostanziale è nell'allocazione di tali oggetti. Quando si crea un oggetto tramite il tag jps:useBean l'oggetto è memorizzato come attributo dell'oggetto di scope e vi si può fare accesso anche tramite i tag di scriptlet. Gli oggetti creati tramite tag di dichiarazioni o scriptlet sono memorizzati come variabili della classe **JSPServlet** e vi si può accedere solo tramite suddetti tag.

Viadiamo che valori può assumere l'attributo scope:

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

- **WebContext:** contesto in cui vengono eseguiti i componenti Web: comprende i parametri di inizializzazione, le risorse associate al contesto Web, gli attributi, le utilità per il log;
- **Session:** variabili di sessione;
- **Request:** l'oggetto corrisponde alla richiesta HTTP;
- **Page:** nella JSP in cui è stato creato l'oggetto.

9.4. **Etichette personalizzate**

Anche i JavaBeans, però, presentano dei problemi. Ad esempio, i nomi dei metodi devono seguire convenzioni piuttosto rigide dando luogo a volte a nomi lunghi, complicati e alla fin fine poco chiari. Per passare degli argomenti ai metodi occorre comunque ricorrere agli scriptlet.

Un'ulteriore proposta di soluzione più flessibile dei JavaBeans è rappresentata dalle etichette personalizzate.

Tag libraries: classi che implementano l'interfaccia **Tag**, e, di solito, estendono le classi **TagSupport** o **BodyTagSupport.**

In molti casi, le etichette personalizzate rappresentano un'alternativa all'uso dei JavaBeans, ma questi ultimi non possono manipolare il contenuto della pagina, e comunque il loro uso richiede una qualche conoscenza di Java.

Le etichette personalizzate sono piuttosto complicate da implementare e hanno un ciclo di vita abbastanza complesso.

Le JavaServer Pages Standard Tag Libraries (JSTL) permettono di gestire oggetti in ambiti di definizione diversi, di iterare su collection, di eseguire test condizionali, di fare parsing e formattazione dei dati, etc.

Le scriptlets rappresentano, almeno per applicazioni di dimensioni modeste, una soluzione più immediata da implementare.

10) Session Tracking

Questo capitolo è dedicato alle soluzioni standard adottabili per la soluzione di un problema tipico delle applicazioni web, riconducibile al fatto che HTTP è un protocollo senza stato: **il session tracking**.

Prima di cominciare, cos'è una sessione?
Una sessione è uno scambio interattivo di informazioni, o meglio, una conversazione tra due o più dispositivi di comunicazione, per esempio un browser e un server web.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Quando c'è una serie continua di richieste e risposte da uno stesso client verso un server web, il server a priori non può identificare il client che ha inviato la richiesta, questo perché HTTP è senza stato. Potrebbe però essere necessario mantenere lo stato della conversazione tra client e server, è qui che entrano in campo le varie tecniche di session tracking.

Sono cinque i metodi per effettuare il session tracking:

1. Richiedere un'autorizzazione all'utente;
2. Campi nascosti nelle form;
3. Riscrittura dell'URL;
4. Cookies;
5. Variabili di sessione.

10.0.1. **Autenticazione richiesta all'utente**

Viene implementata a livello di HTTP, configurando opportunatamente il server. Tramite le variabili di sessione è anche possibile implementare un meccanismo di autenticazione che però risulta molto meno sicuro e quindi sconsigliabile.

Finchè il browser rimane aperto, username e password vengono salvati sul client, in modo che l'utente non debba digitarli tutte le volte che viene inoltrata una richiesta.

Il concetto è che l'utente debba essere autorizzato ad usare specifiche risorse contenute nel server web. Quando giunge una richiesta per una di queste risorse il server risponde con codice di stato 401 (Authorization required) e richiede all'utente di identificarsi inserendo uno username e password. In questo modo il client che ha inviato la richiesta viene identificato e la sessione può essere mantenuta.

Il metodo **getRemoteUser()** fornisce alla servlet lo username dell'utente dopo che questo ha fatto il login fornendolo insieme ad una password.

I vantaggi di questa tecnica sono:

- Facile da implementare: il server protegge determinate pagine e può poi identificare ogni client;
- Il meccanismo delle autenticazioni funziona anche se l'utente accede al sito da macchine diverse;
- Continua a funzionare anche se l'utente esce dal sito o addirittura esce dal browser prima di tornare a concludere la transazione.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Invece gli svantaggi sono:

- La richiesta di registrazione si adatta solo a determinate situazioni (transizioni commerciali, informazioni sensibili); in tutti gli altri casi occorre un session tracking anonimo.
- Un utente può, in ogni momento, mantenere attiva una sola sessione dato un sito.

10.0.2. **Campi nascosti nelle form**

HTML, permette di dichiarare dei campi di input nascosti. Questi campi, non direttamente visibili all'utente, possono essere inseriti in pagine HTML e l'informazione contenuta inviata al server web per il session tracking.

Questo tipo di meccanismo non richiede nessuna configurazione, ne da parte del client, ne da parte del server.

I vantaggi di questa tecnica sono:

- I campi nascosti sono supportati da tutti i browser più diffusi;
- Non richiedono particolari funzionalità del server;
- Sono anonimi, completamente trasparenti all'utente.

Invece gli svantaggi sono:

- Funziona solo quando la sessione prevede una sequenza di schede da riempire generate dinamicamente. Qual'ora la sessione preveda l'invio di pagine statiche questo metodo non può essere adottato.
- Fallisce in caso di errore che provoca la chiusura del browser.

10.0.3 **Riscrittura dell'URL**

Quando viene effettuata una richiesta vengono aggiunti parametri addizionali all'URL, in generale un identificativo di sessione (per esempio un numero univoco). Sarà perciò sufficiente per il server tracciare questo identificativo.

Abbiamo tra metodi per riscrivere un URL:

1. http://server:porta/servlet/Rewritten - **originale;**
2. http://server.porta/servlet/Rewritten/123 - **path aggiuntivo;**
3. http://server.porta/servlet/Rewritten?sessionid=123 – **parametri aggiunti;**
4. http://server.porta/servlet/Rewritten;$sessionid$123 – **cambiamento personalizzato**

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Il modo per implementare ciascuno dei seguenti metodi è grosso modo la stessa. Il client invia una richiesta al server, il server per esempio estrapola dalla richiesta il parametro **sessionid**, qual'ora il parametro sia inesistente provvede a creare un identificativo di sessione. Nel generare la pagina HTML da inviare al client provvederà ad inserire questo identificativo di sessione nell'URL a cui una form fa riferimento. In questo modo una successiva sottomissione di quella form farà giungere al server il parametro **sessionid** opportunatamente settato.

Ecco un esempio con una servlet Java:

```
//il server riceve la richiesta del client

String sessionid = request.getParameter("sessionid");
    if(sessionid == null) {
      sessionid = generateSessionId();
    }

//riscrittura dell' URL

out.println("<form action=\"/servlet/ShoppingCart?sessionid=" + sessionid );
```

Andiamo ad esaminare vantaggi e svantaggi di ciascun metodo:

- **path aggiuntivo**
  - vantaggi
    - funziona su tutti i server;
    - può essere usato come target per form che usano il metodo **POST** o **GET**
  - svantaggi
    - genera problemi nel caso in cui una servlet deve usare il path aggiuntivo come path vero e proprio
- **parametro aggiuntivo**
  - vantaggi
    - funziona su tutti i server;
  - svantaggi
    - non può essere usato per form che usano il **POST**;
    - può causare collisioni nei nomi dei parametri
- **cambiamento personalizzato**
  - vantaggi
    - funziona su tutti i server che supportano tale cambiamento;
    - si possono referenziare documenti statici senza perdere la sessione;

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

- svantaggi
  - non funziona per i server che non supportano tale cambiamento

10.0.4. **Cookies**

Come abbiamo già visto vengono implementati dal protocollo HTTP. Un **cookie** è un pezzo di informazione testuale mandata dal server Web al browser, che può in seguito venire ripassata dal browser al server.

Ogni qualvolta un browser invia una richiesta a un server Web, include in quest'ultima i cookies settati da quello specifico server Web. In questo modo il server, estrapolando i cookies dalla richiesta ricevuta, può identificare il client.

È uno dei modi più utilizzati e facili da implementare per il session tracking.

Per ovvi motivi di sicurezza i browser inviano i cookies esclusivamente al server Web che ne ha richiesto la memorizzazione. L'unico svantaggio nell'usare i cookies è che il browser potrebbe essere impostato per non accettarli. In questo caso l'utente non potrà accedere all'applicazione web che ne richiede l'uso.

Java, e quindi le servlet, offrono una classe **<<tt>Cookie</tt>** con:

- **Costruttore:** public Cookie(String name, String value), dove il nome identifica il cookie, mentre il valore rappresenta l'informazione associata col cookie.
- **Per mandare un cookie da una servlet ad un client:** public void HttpServletResponse.addCookie(Cookie cookie): altri cookies possono essere aggiunti con lo stesso metodo, ma prima di ogni altro contenuto.
- **Per estrarre i cookies da una richiesta di un client una servlet usa**: public Cookie[] HttpServeltRequest.getCookie().

Possiamo quindi ricordare che il problema più importante coi cookies è che non sempre il browser li accetta.

Per default, i cookie vengono eliminati alla fine della sessione di browsing. Per importare una durata maggiore, si usa il metodo setMaxAge, che dà il numero di secondi dopo i quali il cookie verrà eliminato.

Importantissimo: i browser inviano i cookies esclusivamente allo stesso dominio memorizzato nel cookie.

10.0.5. **Le variabili di sessione**

Sono degli oggetti che vengono creati e memorizzati sul server e associati univocamente a ogni client. Possono contenere attributi o riferimenti a altri oggetti

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

che descrivono le precedenti attività del client sul server (per esempio la data dell'ultima connessione del client).

Ogni variabile di sessione ha un identificativo, client e server si scambiano questo identificativo tramite cookies o riscrittura dell'URL.

È uno dei migliori modi di implementare il session tracking. Lo svantaggio è il carico computazionale per il server, soprattutto se deve memorizzare una gran quantità di variabili di sessione.

11) Cascading StyleSheets o CSS

11.1. **Cos'è CSS?**

CSS (Cascading StyleSheets) è un linguaggio di stile di presentazione (style sheet language) usato per descrivere l'aspetto e la presentazione di un documento scritto in un linguaggio di markup.

Cos'è un linguaggio di stile di presentazione?
È un linguaggio di programmazione che esprime la presentazione di documenti strutturati.

CSS in generale viene associato a un documento HTML o XML per descriverne uno stile di presentazione. Lo fa attraverso dei **fogli di stile** con lo scopo di adattare i contenuti alla varietà dei media disponibili.

C'è da dire che è in grado di supportare diversi media come stampanti, video, palmari, braille, ecc.

Applica un principio fondamentale, quello della **separazione tra contenuto e presentazione**.

Il CSS non ha il tradizionale sistema di versioni, ma un'organizzazione a **livelli**. Ogni livello del CSS costruisce sul livello precedente, introducendo rifiniture nelle definizioni e aggiungendo nuove features. Ne deriva che l'insieme delle feature di ogni livello è un soprainsieme di quello del livello immediatamente inferiore, visto che nuove feature possono venir aggiunte ad ogni livello.

L'insieme dei comportamenti ammessi è un sottoinsieme di quelli ammessi nei livelli inferiori, visto che ogni comportamento può venir ulteriormente raffinato da un livello superiore.

Uno user agent che rispetti un livello superiore di CSS è garantito rispettare anche quelli inferiori.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

11.1.1. **Le regole**

Un documento CSS è un insieme di regole che associano a ogni elemento di un file HTML, per esempio delle proprietà che ne definiscono uno stile di formattazione (per esempio di che colore deve appararire una scritta).

Ogni regola è composta da due elementi:

1. Un **selettore**, che identifica l'elemento all'interno del documento a cui applicare la regola;
2. Una **dichiarazione**, che descrive la presentazione del blocco identificativo dal selettore.
   Ogni dichiarazione è composta da:
   a. Una **proprietà**, che identifica una particolare caratteristica dell'elemento;
   b. Un **valore**, separato dalla proprietà dai ":" che infine ne descrive il metodo di visualizzazione

Facciamo un esempio, consideriamo questa regola applicata a un documento HTML.

```
H1 {color : blue}
```

In questa regola stiamo dicendo che il testo racchiuso in tutti gli elementi di tipo "H1", all'inteno del documento HTML, sarà visualizzato in blu.

- "H1" è il selettore;
- "{color : blue}" è il blocco dichiarativo;
- "color" è la proprietà;
- "blue" è il valore della proprietà color.

Tutti e quattro insieme compongono una regola CSS.

È possibile tramite i **selettori** identificare in maniera molto specifica a quali elementi applicare la regola seguente. Per esempio la regola così scritta "**E > F {color : black }**" si applica a tutti gli elementi di F che sono figli di un elemento.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Di seguito una lista completa per riferimento:

*	qualunque elemento
E	qualunque elemento di nome E
E, F	qualunque elemento E e qualunque elemento F
E F	qualunque elemento F dentro/sotto un E <E> ... <X> ... <F> </F> .... </X> </E>
E > F	qualunque elemento F figlio di un E
E + F	qualunque elemento F preceduto da un E <E> ... </E> <F> ... </F>
E[attr]	qualunque elemento E con l'attributo attr impostato
E[attr="x"]	qualunque elemento E con attr uguale a x
E[attr ="x"]	qualunque elemento E con attr uguale ad una lista di valori separati da spazi, che include x

Esistono poi gli **pseudo-elementi** che corrispondono a delle pseudo-etichette che consentono all'autore del documento di modificare specifiche parti un dato elemento:

E:first-child ovvio <X> <E> ... </E></X> , si prende il primo elemento
E:link per sorgenti di hyperlink non visitati
E:visited per sorgenti di hyperlink già visitati
E:active elementi selezionati col mouse
E:hover elementi su cui si trova il mouse
E:focus elementi che hanno il focus

11.1.2. **Conflitti e loro risoluzione**

CSS prescrive 3 tipi di stilesheets:

1. Quello definito dall'autore del documento
2. Quello definito dall'utente
3. Quello definito come default dallo user agent

Nel caso di conflitti, la regola di soluzione prevede che autore > utente > stile di default dello user agent.

Overriding di regole ereditate : regole esplicite > proprietà ereditate.

11.1.3. **Fogli di stile dipendenti dai tipi di media**

È possibile condizionare il foglio di stile da applicare ai diversi **tipi di media**. Ad esempio, un documnto potrebbe adottare un font sans-serif quando mostra a video e serif quando stampato.

"screen" e "print" sono due tra i tipi di media previsti, probabilmente quelli usati più di frequente.

Ci sono due sistemi per condizionare il foglio di stile al tipo di media:

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

1. Dal foglio di stile, con le regole **@media:** @media print {background : white; color : black}

```
@media print {
    BODY { font-size: 10pt }
}
@media screen {
    BODY { font-size: 12pt }
}
@media screen, print {
    BODY { line-height: 1.2 }
}
```

2. Dal documento, ad esempio, specificando un attributo media nell'elemento **link dell'HTML 4.0.**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
  <HEAD>
     <TITLE>Link to a target medium</TITLE>
     <LINK rel="stylesheet" type="text/css"
     {\bf media="print, handheld"} href="foo.css">
   </HEAD>
   <BODY>
      <P>Il contenuto della pagina ...
   </BODY>
</HTML>
```

Le **media queries** estendono le funzionalità dei tipi di media permettendo una classificazione dei fogli di stile ancora più precisa, visto che oltre al tipo di media si possono specificare anche delle espressioni che controllano se sono verificate delle **media features**. Si possono controllare i valori di **width, height** e **color**.

12) Programmazione lato client: JavaScript

JavaScript è un linguaggio di scripting orientato agli oggetti, usato comunemente nell'ambito di applicazioni web come strumento di programmazione lato client.

Cos'è un linguaggio scripting?
Sono linguaggi di programmazione che supportano la scrittura di scripting, programmi scritti per essere eseguiti in un ambiente software esistente e che consentono di automatizzarne alcuni compiti.

Sono linguaggi interpretati, non vengono compilati.

Il sistema esistente fornisce un ambiente ospite che comprende oggetti e funzionalità completando in questo le capacità del linguaggio di scripting.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

JavaScript è stato ideato per essere un linguaggio di scripting per il web, sia lato client sia lato server. Il codice JavaScript va immerso nel codice HTML di una pagina attraverso il tag script. Il codice dello script va inserito all'interno del tag, oppure tramite l'attributo source si può fare riferimento a un file su server.

Possono essere inclusi ovunque nella pagina, solitamente nella head vanno gli script che verranno richiamati da azioni dell'utente, nel body invece quelli da eseguirsi al caricamento della pagina. Il nome JavaScript è fuorviante, infatti non ha niente a che fare con Java, il nome è stato deciso per questioni relative al marketing.

Vediamo le principali differenze tra Java e JavaScript:

Java	**JavaScript**
Usato per scopi differenti	Usato esclusivamente per programmazione Web
Codice compilato	Codice non compilato
Codice compilato in applets separate da HTML	Codice immerso nelle pagine HTML
Linguaggio di programmazione	Linguaggio di scripting
Fortemente tipato	Debolmente tipato (type-checking a run-time o assente)
Controllo di accesso a più livelli	Controllo di accesso semplice
Rigida verifica degli array	Nessuna verifica
Gerarchie di classe	Gerarchie di istanze

JavaScript è un linguaggio basato ad oggetti e guidato dagli eventi (event driven). E' un agglomerato di oggetti in comunicazione tra loro, gli oggetti sono forniti dal linguaggio e dall'ambiente host. Un browser Web per esempio fornisce a JavaScript un ambiente di esecuzione per l'elaborazione lato client di oggetti che rappresentano finestre, menu, pop-up, ecc… e inoltre metodi per allegare codice JavaScript a determinati eventi.

I browser implementano diversi meccanismi di sicurezza per quanto riguarda gli script JavaScript.

In particolare uno script JavaScript non può eseguire di nascosto le seguenti operazioni:

- Impostare e recuperare i parametri delle preferenze del browser, le funzioni di aspetto della finestra principale, i pulsanti di azione e di stampa;
- Eseguire un'applicazione sul computer client;
- Leggere o scrivere file o directory sul computer del client;

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

- Catturare in presa diretta flussi di dati dal server per ritrasmetterli.
- Inviare al programmatore messaggi segreti di posta elettronica dai visitatori del sito Web

Andiamo a studiare un po' la sintassi e le caratteristiche di JavaScript.

Variabili, operatori, funzioni e strutture di controllo:
tramite la parola chiave **var** si possono dichiarare variabili, non occorre specificarne il tipo. Quando una variabile viene dichiarata all'esterno di una funzione ha l'ambito di una **variabile globale**, tutta la pagina HTML.

Altri script definiti in altri tag possono fare riferimento a tale variabile! Bisogna quindi porre molta attenzione sulla scelta dei nomi e alle dichiarazioni. I **parametri** di una funzione vengono considerati come **variabili locali**.

Per quanto riguarda gli **operatori**, sono più o meno gli stessi che si travano nei più comuni linguaggi di programmazione. Stessa cosa per le strutture di controllo (if then else, for, while, do-while, ecc.)

Per quanto riguarda le **funzioni** invece, è buona norma **dichiararle nell'intestazione** in modo da rendersi disponibili in qualsiasi punto. Possono ritornare un valore tramite l'istruzione **return**, non può esserci overloading, in quanto possono essere passati in ingresso a una funzione anche più argomenti di quelli previsti. Il **passaggio di parametri** avviene per **riferimento se si tratta di oggetti**, per **valore per gli altri tipi di dati**.

JavaScript mette a disposizione alcuni tipi primitivi:

1. **Undefinded**, corrisponde ad un solo valore (valore undefinded appunto), è il tipo di una variabile che non è ancora stata inizializzata;
2. **Null**, tipo primitivo di un oggetto ancora non referenziato (come undefinded ha un solo valore il valore null);
3. **Boolean**, tipo booleano (valori assumibili **true** e **false**);
4. **Number**, tipo numerico (rappresentazione in floating point a 64 bit) include anche i valori di infinito positivo e negativo a **NaN** (Not a Number);
5. **String**, tipo stringa;
6. **Object**, per la definizione di oggetti.

Un oggetto in JavaScript è un insieme di proprietà, una proprietà può essere un altro oggetto, una funzione o un tipo primitivo. Esistono due categorie di oggetti, oggetti nativi che sono quelli messi a disposizione dal linguaggio (built-in) e quelli costruiti dal programmatore e gli oggetti host che sono quelli messi a disposizione dall'ambiente ospite.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Gli oggetti **built-in** includono gli oggetti **Global, Object, Function, Array, String, Boolean, Number, Math, Date, RegExp** e gli oggetti per la gestione degli errori **Error, EvalError, RangeError, ReferenceError, SyntaxError, TypeError** e **URIError.**

Attenzione a non confondere il tipo primitivo string con l'oggetto built-in String!

JavaScript prevede un **costruttore** per ogni oggetto built-in e host. Attributi e metodi possono essere aggiunti all'oggetto di base. Ogni oggetto deriva da Object, anche se JavaScript non implementa una vera ereditarietà.

Per gli oggetti definiti dall'utente è necessario specificarne un costruttore. Proprietà (attributi e metodi) possono essere aggiunti a un oggetto anche al di fuori del costruttore!

Vediamo come viene implementata l'ereditarietà in JavaScript. Si basa sui prototipi e non sul concetto di classe come in Java. Ad ogni costruttore è associata una proprietà prototype usata per implementare l'ereditarietà di struttura, stato e comportamento.

Ci si riferisce al prototipo associato al costrutture tramite l'espressione **<costruttore>.prototype**. Ogni proprietà aggiunta al prototipo di un costruttore viene condivisa da tutti gli oggetti che condividono quel prototipo. Un prototipo può avere un riferimento ad un altro prototipo, ecco che si crea una sorta di ereditarietà.

Ogni volta che si chiama una proprietà su un oggetto succede che:

- La proprietà viene cercata sull'oggetto stesso, se esiste viene restituita;
- Se non c'è, viene cercata a partire dal prototipo;
- Viene cercata risalendo la catena dei prototipi finche non viene trovata o la catena si interrompe (riferimento null)

In sostanza in JavaScript attraverso il costruttore si definisce e si creano insiemi di oggetti, per costruire una gerarchia di oggetti si assegna al protoipo del costruttore un altro oggetto. Per esempio:

```
function Impiegato(){
  this.name = “”;
}

function Manager(){
  this.reparto = “general”;
}

Manager.prototype = new Impiegato;
```

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Se vogliamo aggiungere una proprietà a un oggetto e vogliamo che sia condivisa da tutti, dobbiamo aggiungerla al prototipo, per esempio:

```
mark = new Manager;
mark.name = “mark”;
mark.reparto = “amministrazione”;
mark.bonus = 3000;
```

ecco che abbiamo aggiunto la proprietà “bonus” all’oggetto mark. Ma se vogliamo che tutti gli impiegati abbiano una proprietà “bonus” allora dobbiamo scrivere:

```
Impiegato.prototype.bonus = “”;
```

13) Document Object Model, DOM

13.1 **Cos’è il DOM?**

Il DOM è un’API, independente dalla piattaforma, che descrive la struttura di un documento HTML o XML, tramite il quale i programmatori possono accedere e modificare tutti gli elementi del documento stesso.

Quando JavaScript viene usato lato client, gli oggetti dell’host devono permettere al linguaggio di compiere tutte le elaborazioni richieste lato client, e quindi di elaborare il documento da presentare all’utente e tutti gli eventi ad esso collegati. Occorre quindi una struttura dati che rappresenti tale documento.

La DOM rappresenta la risposta a questa esigenza: l’acronimo sta per **Document Object Model**.

La struttura logica di un documento XML o HTML è ad albero: ogni nodo corrisponde ad un elemento e i figli di ogni nodo corrispondono ai sottoelementi dell’elemento corrispondente.

Tale albero rappresenta una **gerarchia di contenimento**, non di ereditarietà:

- Nessun oggetto eredita proprietà o metodi da un oggetto superiore nella catena;
- Non esiste un passaggio automatico di messaggi da un oggetto all’altro in nessuna direzione.

È molto importante sottolineare che il DOM è indipendente dalla piattaforma, è infatti un’interfaccia definita dal W3C. Non dipende dal browser usato ne dal

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

sistema operativo. Questo significa però che alcuni browser non implementino un DOM proprietario e differente dal DOM W3C.

In sostanza il DOM è un modello che descrive come i diversi oggetti di una pagina sono collegati tra loro.

Un browser può supportare o no il DOM W3C, se lo supporta significa che da una sua implementazione dell'interfaccia definita dal DOM W3C. **Pertanto la DOM viene implementata dal browser, il W3C definisce l'interfaccia. I vari linguaggi di scripting lato client come JavaScript o VBscript sono solo degli strumenti con i quali accedere alla DOM, non ne danno alcuna implementazione!**

Gli oggetti documento predefiniti sono generati solo quando viene caricato nel browser il codice HTML contenente le loro definizioni. I browser compatibili con JavaScript creano in memoria gli oggetti a partire dal codice HTML man mano che questo viene caricato, indipendentemente dal fatto che gli script li usino o meno.

Gli oggetti vengono creati nell'ordine in cui vengono caricati: dopo aver creato un ambiente multiframe, uno script non può comunicare con gli oggetti di un altro frame fino a quando entrambi i frame non sono caricati.

Proprietà: numeri, stringhe, ma anche array e, naturalmente, metodi. Sensibili alle maiuscole in JavaScript, ma non in HTML. Alcune proprietà sono accessibili solo in lettura, mentre per altre è possibile cambiarne il valore.

Un **gestore di eventi** specifica la reazione di un oggetto a un evento attivato da un'azione dell'utente (click,..) o del browser (caricamento di un documento,…).

13.3 **Il DOM W3C**

Il DOM (Documento Object Model) del W3C si propone come un'interfaccia indipendente sia dalla piattaforma che dal linguaggio che permette ai programmi e script l'accesso e l'aggiornamento dinamico di contenuti, struttura e stile dei documenti: **si tratta quindi di un API.**

Permette quindi di elaborare il documento incorporando i risultati di tale elaborazione nel documento stesso.

Il DOM permette ai programmatori di costruire documenti, navigare all'interno della loro struttura e aggiungere, modificare o cancellare elementi e contenuti.

Il suo scopo è di rendere possibile per i programmatori scrivere applicazioni che lavorino adeguatamente su tutti i browser e server e su tutte le piattaforme.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

L'architettura della DOM, è divisa in moduli ciascuno dei quali si riferisce a un dominio. Andiamo a esaminare questi moduli:

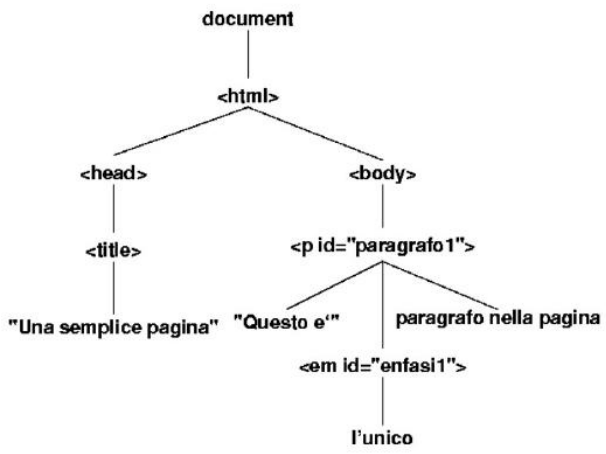
- **core DOM:** specifiche per la struttura del documento, condivisa dai documenti HTML e XML;
- **XML DOM:** estende il modulo **core DOM** per le necessità di XML 1.0;
- **HTML DOM:** come prima estende la **core DOM**, rivolgendosi in più alle caratteristiche di HTML;
- **eventi DOM:** definisce gli eventi orientati alla modifica nell'albero XML attraverso eventi generati dall'utente (quali click del mouse o della tastiera) o specifici di HTML;
- **DOM CSS:** orientata ai documenti CSS.
- **DOM Load and Save:** l'esigenza di caricare un documento XML in un albero DOM e viceversa di salvare un albero DOM in un documento XML è fondamentale.
- **DOM Validation:** modifica l'albero DOM mantenendone la validità;
- **DOM XPath**

Qualsiasi elemento si intenda usare nell'elaborazione deve essere contraddistinto da un identificatore assegnato all'attributo ID.

Gli identificatori univoci rendono molto più facili i riferimenti agli oggetti e non sono influenzati da eventuali modifiche all'ordine degli elementi.

Nel DOM, i documenti hanno una struttura logica che assomiglia molto ad un **albero**, precisamente una foresta o boschetto, che può contenere uno o più alberi. Si tratta di un **modello logico**, che può venir implementato in qualsiasi modo risulti più conveniente.

```
<html>
  <head>
    <title> Una semplice pagina </title>
  </head>
  <body>
    <p id="paragrafo1"> Questo e' <em id="enfasi"> l'unico </em> paragrafo della pagina. </p>
  </body>
</html>
```



Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

I concetti introdotti dal DOM W3C, sono nuovi modi per fare riferimento ad elementi e nodi presenti in un documento. L'oggetto **document** viene messo a disposizione automaticamente dal browser quando un documento viene caricato. Esso racchiude tutti gli elementi (con loro contenuto e attributi) presenti nel documento.

La DOM associa a ogni documento una struttura logica ad albero, il nodo radice è identificato dall'oggetto **document**. Ciascun contenitore, che sia un elemento o solo parte di testo rappresenta un nodo all'interno dell'albero. Per ogni nodo vengono definite delle proprietà, alle quali accedere tramite funzioni, e dei metodi con i quali modificare o aggiornare la struttura di un nodo.

Il DOM W3C fino al secondo livello prevede 12 tipi di nodi, di cui 7 si applicano ai documenti HTML:

TIPO	NUMERO	NOMENODO	VALORENODO	DESCRIZIONE
Elemento	1	nome tag	null	Qualsiasi elemento HTML con tag
Attributo	2	nome attributo	valore attributo	Una coppia nome-valore attributo in un elemento
Testo	3	#text	contenuto di testo	Un frammento di testo contenuto in un elemento
Commento	8	#comment	testo di commento	Commento HTML
Documento	9	#document	null	Oggetto document radice
TipoDocumento	10	DOCTYPE	null	Specifica DTD
Frammento	11	#document-fragment	null	Serie di uno o più nodi fuori dal documento

Nel primo esempio visto sopra, si avevano 1 nodo documento (tipo 9), 6 nodi elemento (tipo 1) e 4 nodi di testo (tipo 3).

Proprietà dei nodi

Proprietà	Valore	Descrizione
nodeName	string	Varia col tipo di nodo
nodeValue	string	Varia col tipo di nodo
nodeType	number	Costante corrispondente a ciascun tipo
parentNode	object	Riferimento al contenitore immediatamente più esterno
childNodes	array	Tutti i nodi figli nell'ordine in cui si trovano nel sorgente
firstChild	object	Riferimento al primo nodo figlio
lastChild	object	Riferimento all'ultimo nodo figlio
previousSibling	object	Riferimento al precedente fratello
nextSibling	object	Riferimento al successivo fratello
attributes	MappaNodo	Array di nodi attributi
ownerDocument	object	Riferimento al documento contenitore che funge da radice
namespaceURI	string	URI alla definizione dello spazio dei nomi (solo nodi elemento e attributo
prefix	string	prefisso nella definizione dello spazio dei nomi (solo nodi elemento e attributo

METODO	DESCRIZIONE
appendChild(nuovoFiglio)	Aggiunge un nodo figlio alla fine del nodo corrente
cloneNode(profondità)	Prende una copia del nodo corrente (opzionalmente con figli)
hasChildNodes()	Determina se il nodo corrente ha figli (booleano)
insertBefore(nuovo,rif)	Inserisce un nuovo figlio davanti ad un altro figlio
removeChild(vecchio)	Cancella un figlio
replaceChild(nuovo, vecchio)	Sostituisce un vecchio figlio con uno nuovo
supports(funzione,versione)	Determina se il nodo supporta una determinata funzione

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

13.3.2. **Attributi**

Gli attributi associati ai tag di elemento vengono convertiti dal browser in proprietà dell'oggetto. Vale a dire che le linee due e tre del seguente codice JavaScript hanno lo stesso effetto.

```
var table = document.getElementById("mytable");
table.setAttribute("border", 2);
table.border = 2;
```

- **getAttribute(string name):** torna il valore dell'attributo name;
- **setAttribute(string name, string value):** se c'è già un attributo con quel nome, ne cambia il valore, altrimenti aggiunge l'attributo

13.3.3. **Elementi Style**

Ogni regola CSS per un elemento è rappresentato come una proprietà dell'oggetto corrispondente all'elemento style. Continuando il codice di prima potrei scrivere:

```
table.style.backgroundColor = "codice RGB";
var color = table.style.backgroundColor;
```

Ovviamente, le proprietà che non sono state inizializzate sono vuote.

Poiché i nomi delle proprietà non possono contenere il trattino, occorre convertire le proprietà delle regole CSS in proprietà del DOM da elaborare con JavaScript.

Vengono tradotte in questo modo:

CSS property	**JavaScript equivalent**
background-color	style.backgroundColor
font-size	style.fontSize
left	style.left
border-top-width	style.borderTopWidth

13.3.4. **Gestione degli eventi**

Vediamo adesso il modulo della DOM che si occupa degli eventi. Il modulo in esame permette la registrazione di rilevatori di eventi (**event listeners)**, descrivere il flusso degli eventi tramite una struttura ad albero, mantenere la compatibilità con il sistema di eventi del browser.

Con il termine evento indichiamo la rappresentazione di qualcosa che avviene in modo asincrono, per esempio come il click del mouse. Possiamo dividere gli eventi in tre grandi insiemi:

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

- **eventi UI**: riguardanti l'interfaccia utente. Generati dall'interazione dell'utente con l'interfaccia attraverso dispositivi esterni come mouse o tastiera;
- **eventi logici UI**: indipendenti dal tipo di dispositivo, per esempio un cambio di focus;
- **eventi di mutazione**: causati da un'azione che modifica la struttura del documento.

Ogni evento è associato ad un oggetto, il suo event target. Il rilevatore di eventi è un meccanismo che dice ad un oggetto di rispondere ad un particolare evento.
Supponiamo di avere un pulsante per attivare una funzione di calcolo:

document.getElemetById("calcButton").addEventListener("click", doCal, false)

In questo modo ad ogni click del mouse sul bottone **calcButton** verrà mandata in esecuzione la funzione **doCalc**. Tuttavia per quanto riguarda gli eventi bisoga tenere presente che:

- **Flusso di propagazione degli eventi**: processo secondo il quale un evento ha origine nell'implementazione del DOM e viene passato all'interno del DOM;
- **Target dell'evento**: nodo verso il quale l'evento è diretto (proprietà **target** del nodo **Event**);
- Tutti gli **EventListener** sul target vengono attivati ma l'ordine però non è specificato;
- Un **EventListener** può bloccare l'ulteriore propagazione dell'evento chiamando il metodo **StopPropagation** dell'interfaccia **Event** oltre il nodo corrente, sia in disesa che in salita (ma gli **EventListener** registrati su quel nodo ricevono l'evento).
- Durante la fase di bubbling, vengono attivati tutti gli **EventListner** trovati fino al document compreso, ma escludendo gli **EventListener** registrati per la cattura;
- Anche se l'albero viene modificato durante l'elaborazione dell'evento, il flusso dello stesso procede seguendo lo stato iniziale dell'albero;
- **Cancellabile:** questo termine indica che gli **EventListener** registrati per l'evento considerato possono cancellare le azioni associate all'evento di default (ad esempio, l'attivazione di un hyperlink).

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

18) AJAX: Asynchronous JavaScript and XML

Asynchronous JavaScript and XML (AJAX) non è di per sé una tecnologia, ma è un termine che descrive un nuovo approccio all'utilizzo di diverse tecnologie esistenti, compresi: HTML o XHTML, CSS, JavaScript, DOM, XML, XSLT e l'oggetto XMLHttpRequest.

Grazie all'utilizzo di queste tecnologie in combinazione con il modello AJAX, le applicazioni web possono eseguire aggiornamenti rapidi e **incrementarli** dell'interfaccia utente senza ricaricare nel browser l'intera pagina. Questo rende l'applicazione più reattiva alle azioni dell'utente.

Nelle classiche applicazioni web una richiesta HTTP viene preparata e inviata a seguito di una qualche azione dell'utente. Il server web reagisce alla richiesta con un'opportuna risposta, sempre HTTP, a cui il browser reagisce preparando il materiale da presentare all'utente. Questo funzionamento è **sincrono** rispetto alle azioni dell'utente.

A volte basta un'azione lato client, che può ad esempio essere risolta con uno strumento di programmazione lato client, quale JavaScript o una applet. A volte, inoltre, la maggior parte della pagina rimane invariata, e quindi basta aggiornarne solo una piccola parte.

AJAX permette di programmare un'interazione **asincrona** rispetto alle azioni dell'utente, dove una richiesta HTTP viene preparata e inoltrata solo quando risulta necessario per l'applicazione nel suo complesso.

18.2 **Precursori**

Prima di AJAX si usava un trucco da parte del browser: un **frame fittizio** di dimensioni nulle conteneva un form che veniva riempito e sottomesso mediante JavaScript, senza che l'utente se ne avvedesse direttamente. Quando la risposta HTTP tornava al frame nascosto, esso conteneva un altro script JavaScript che comunicava allo script di partenza l'avvenuta ricezione dei risultati.

18.3 **Ruolo dei diversi componenti**

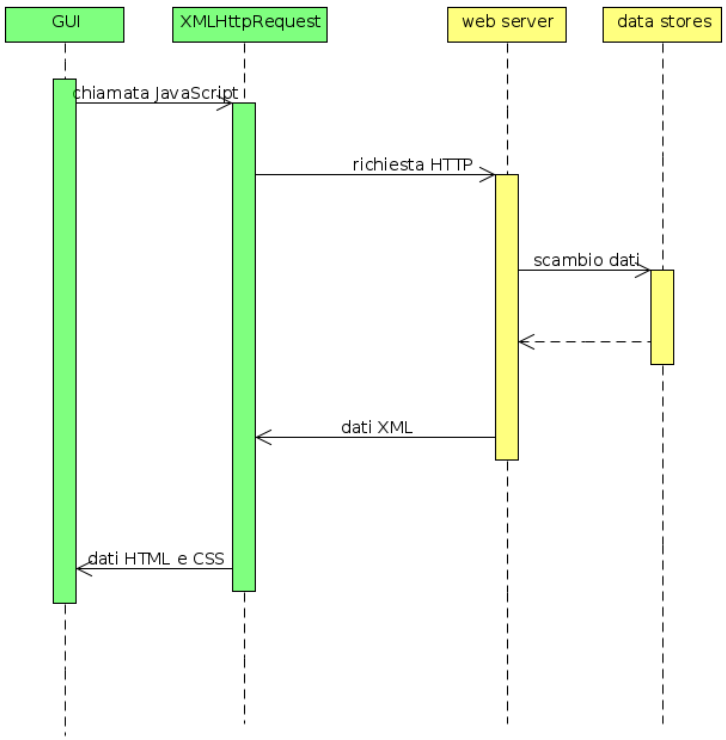
In generale AJAX permette di ottimizzare lo scambio di dati tra client e server andando a richiedere solo i dati necessari a costruire la parte della pagina che va modificata anziché ricaricare ogni volta tutta la pagina.

Il modello AJAX si basa sugli strumenti seguenti:

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

- **JavaScript** per permettere la comunicazione col browser e per rispondere ai diversi eventi;
- **DOM:** per poter accedere e modificare la struttura della pagina HTML o XHTML;
- **XML:** per rappresentare i dati che vengono scambiati tra server e client;
- L'oggetto **XMLHttpRequest** per **permettere uno scambio asincrono** dei dati tra client e server.

In figura un esempio di sequenza con modello AJAX (verde = client / giallo = server)



1. A seguito di un evento generato dall'utente, che ad esempio preme un bottone, viene **eseguita una chiamata JavaScript;**
2. Viene **creato e configurato un oggetto XMLHttpRequest**; in particolare viene impostato un parametro di richiesta che include l'identificativo del componente che ha generato l'evento e qualsiasi valore immesso dall'utente;
3. L'oggetto **XMLHttpRequest manda al server una richiesta HTTP in modo asincrono rispetto all'azione dell'utente**. Il **server web elabora la richiesta** come farebbe normalmente (ad esempio, attraverso un CGI, o una servlet) e, se necessario, salva i dati passati nella richiesta in modo persistente;
4. Il **server web prepara quindi una risposta HTTP** nel cui body viene inserito un **documento XML** con un tutti gli aggiornamenti che sono stati richiesti dal client;
5. L'oggetto **XMLHttpRequest riceve i dati XML,** li **elabora e aggiorna opportunatamente la DOM** della pagina (X)HTML visualizzata dal browser.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

18.3.1 L'oggetto XMLHttpRequest

L'oggetto XMLHttpRequest risulta centrale in tutto il meccanismo, perché rappresenta un'interfaccia che permette ai programmi (o agli scripts) di implementare funzionalità proprie di un client HTTP.

Il W3C ha quindi specificato un minimo insieme di caratteristiche interoperabili basato sulle implementazioni esistenti.

```
//In ECMAScript, si crea un'istanza di XMLHttpRequest usando il costruttore:
var r = new XMLHttpRequest();
```

19) Programmazione su web

Oltre a un web browser per accedere alle informazioni disponibili sul web è interessante studiare come progettare componenti software che possano svolgere compiti in modo automatizzato. Per esempio potremmo pensare a un'applicazione che data una pagina HTML in ingresso verifichi che tutti i link verso altri documenti presenti nella pagina siano validi.

Andiamo a studiare quattro tipi di applicazioni web automatizzate:

- **bot**, un programma che raccoglie informazioni da specifiche locazioni;
- **agent**, simile al bot, ma un utente può specificare il tipo di informazioni da raccogliere;
- **aggregator,** combina l'informazione raccolta, anche da un gran qualità di siti, in modo chiaro e compatto per presentarla all'utente;
- **spider,** un bot ottimizzato per visitare un gran numero di pagine web.

Ci si può riferire con il termine **web crawler** per identificare un bot, un agent o uno spider. Come dice la definizione più generale, un **web crawler** è un programma che esplora il WWW in maniera metodica e automatica. Molti di questi programmi vengono scritti in Java, questo perché Java mette a disposizione, come visto fin ora, un ottimo supporto built-in per il protocollo HTTP e per il parsing di documenti HTML.

I **crawler** sono tipicamente utilizzati dai motori di ricerca per la raccolta di informazioni e per l'indicizzazione di queste ultime.

Vediamo come progettare un crawler. A un crawler è richiesto di raccogliere un gran numero di pagine in modo rapido ed efficiente insieme con la struttura che le interconnette.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

In più deve essere in grado di non essere depistato da eventuali “spider traps”, pagine web preparate per depistare un crawler o bloccarlo in un ciclo infinito e rispettoso delle politiche implicite ed esplicite imposte dagli amministratori di siti web.

Deve essere scalabile, distribuito, ottimizzato in termini di prestazioni ed efficienza, in grado di riconoscere lo SPAM, lavorare in modo continuo mantenendo l’indice aggiornato e organizzato in modo modulare in modo che sia semplice da estendere verso nuovi formati, protocolli, ecc.

Ma come fa un crawler a visitare il web? Dobbiamo farci prima di tutto un idea di come è strutturato il web. Il web è composta da **pagine web**, tipicamente in HTML, le pagine sono collegate tra loro attraverso dei **link**, quindi se consideriamo le **pagine** come dei **nodi** e **link** come degli **archi**, abbiamo che il web risulta essere un **immenso grafo orientato.** Da tenere presente che non si tratta di un grafo fortemente connesso! Esistono pagine che non sono accessibili attraverso dei link in altre pagine, ma solo conoscendone a priori l’URL.

Pertanto il processo di visita del web da parte di un crawler altro non è che la visita di un gigantesco grafo. Per visitare un grafo un crawler ha bisogno di alcuni nodi di frontiera, da cui partire per esaminare il web, la cosidetta **URL frontier.** La URL frontier non è statica, può essere aggiornata e modificata dal crawler.

Tuttavia i produttori di server web mettono a disposizione degli amministratori mezzi per impedire la visita da parte di crawler del proprio sito. Di solito le specifiche vengono inserite in un file chiamato **robot.txt** presente nella directory root del server. È possibile identificare un crawler analizzando la frequenza di accesso di un certo IP oppure la modalità delle richieste per le pagine, per esempio un crawler può richiedere di visualizzare una pagina in un puro testo.

20) Servizi web

20.1 **Web Semantico (dati) e Servizi Web (programmi)**

Il Web Semantico (dati) può essere visto come il tentativo di collegare i **dati** presenti nelle diverse basi di dati, mettendole in relazione con modelli del mondo, le **ontologie**.

L’idea è quella di raccogliere tutti i dati disponibili sul web in una enorme base di dati che oltrepassi i singoli componenti.

I **servizi web (programmi)** rappresentano un mezzo standard per interoperare tra diverse applicazioni software, eseguiti su piattaforme e/o contesti diversi.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

I servizi web poggiano sui seguenti **strumenti**:

- **XML:** per serializzare l'informazione;
- **SOAP:** protocollo su HTTP per lo scambio dei messaggi;
- **WSDL:** per descrivere l'interfaccia in un formato che sia elaborabile automaticamente.

Un **servizio Web** è una nozione astratta che deve essere implementato da un **agente**, ovvero un pezzo di software o hardware che spedisce e riceve messaggi.

I due momenti diversi, lo stesso servizio Web può essere implementato da due agenti diversi, magari realizzati con due diversi linguaggi di programmazione.

- **Provider:** fornisce i servizi;
- **Requester:** (quasi sempre) richiede i servizi; in ogni caso ne utilizza;
- **Entity:** persona o organizzazione;
- **Agent:** programma

Occorre prima di tutto un accordo tra provider entity e requester entity su **semantica** e **meccanismo** dello scambio di messaggi.

Per la descrizione del meccanismo dello scambio dei messaggi si usa un linguaggio per la descrizione dei servizi Web: il **Web Service Description Language (WSDL).**

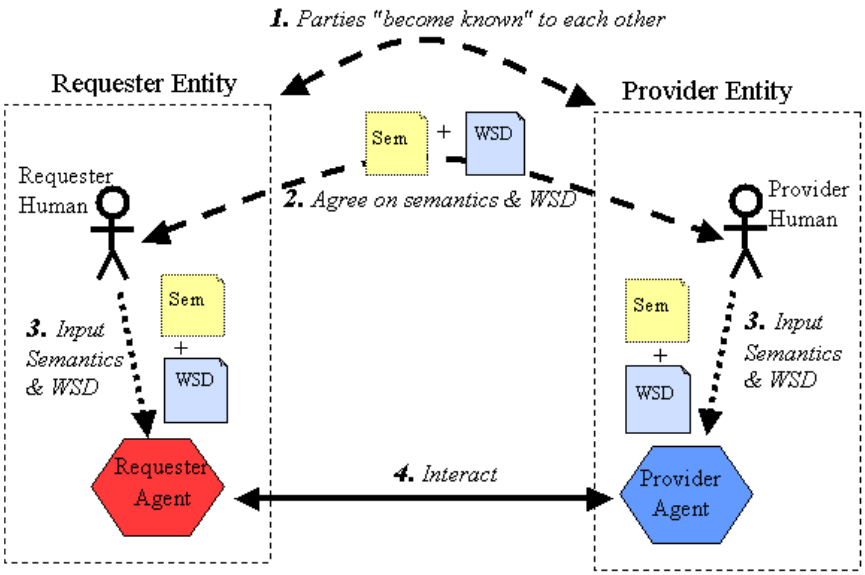
WSDL permette di descrivere il servizio, e rappresenta un accordo che sta alla base dei meccanismi di interazione con quel servizio:

- formato dei messaggi, tipo dei dati, protocolli di trasporto, formati di serializzazione usati nel trasporto tra agente requester e agente provider;
- una o più locazioni in cui un agent provider può venir invocato, dando indicazioni sullo scambio di messaggi che si aspetta.

La **semantica** di un servizio Web è invece data dalle aspettative comuni sul comportamento del servizio, in particolare in risposta ai messaggi inviati. Coinvolge le **entità** più che gli agenti, e non è necessariamente esplicito.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Interazione requester/provider nel caso statico



1. Requester e provider diventano noti l'uno all'altro; due casi:
    a. Inizia l'agente del requester, e deve ottenere l'indirizzo dell'agente del provider, o direttamente dall'entità del provider, o attraverso un discovery service che fornisce l'indirizzo attraverso una descrizione funzionale che può essere esaminata manualmente o automaticamente;
    b. Quando invece (più di rado) a cominciare l'interazione è il provider, esso è venuto a conoscenza dell'indirizzo del requester in un modo che dipende dall'applicazione.
2. Accordo sulla descrizione del servizio (un documento WSDL), non necessariamente esplicito: questo passo può anche precedere il precedente, del tutto o in parte;
3. L'agente del provider e quello del requester devono implementare le funzionalità loro richieste;
4. L'agente del requester e quello del provider si scambiano messaggi SOAP da parte dei rispettivi proprietari

I servizi Web (**SOAP/WSDL**) non sono necessariamente la miglior soluzione per implementare un'architettura orientata ai servizi: **COM** e **CORBA** sono alternative.

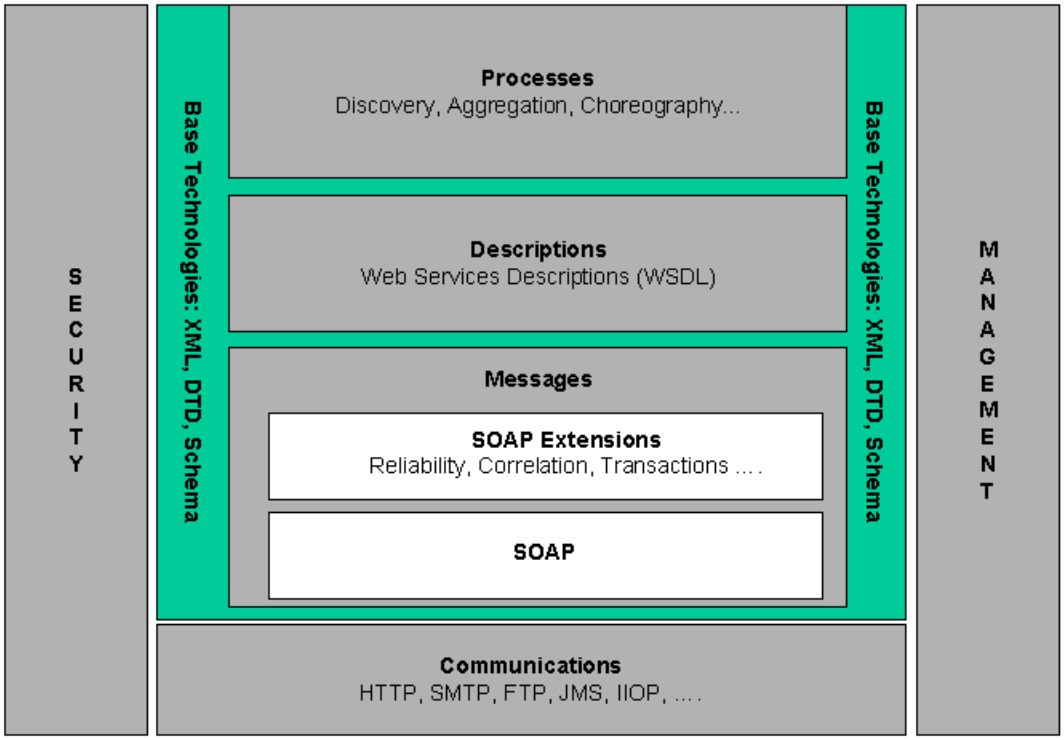
I servizi Web risultano vantaggiosi quando:

- Devono operare in Internet dove affidabilità e velocità non sono garantite;
- Non è possibile garantire che requester e provider siano aggiornati contemporaneamente;
- Si ha una disomogeneità nelle piattaforme su cui vengono eseguite le diversi componenti del sistema distribuito;

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

- Un'applicazione esistente deve essere esposta sulla rete e può venir "impacchettata" come un servizio Web.

SOAP



SOAP **impacchetta e scambia messaggi XML.**

I messaggi SOAP possono venir trasportati da una varietà di protocolli, tra cui HTTP, SMTP, FTP, RMI/IIOP.

SOAP è diventato un nome, e non più un acronimo; tuttavia:

- **Service Oriented Architecture Protocol:** un messaggio SOAP rappresenta l'informazione necessaria per invocare un servizio o riflette il risultato dell'invocazione di un servizio e contiene l'informazione specificata nella definizione dell'interfaccia del servizio.
- **Simple Object Access Protocol:** meccanismo per invocare oggetti remoti, serializzando la lista degli argomenti che deve essere trasportata dall'ambiente locale a quello remoto.

WSDL

Linguaggio per **descrivere** servizi Web.

La descrizione cominicia coi messaggi scambiati tra gli agenti del requester e del provider.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Le definizioni dei servizi Web possono poi venir mappate verso un qualsiasi linguaggio di programmazione, piattaforma, modello di oggetto o sistemi di messaggi.

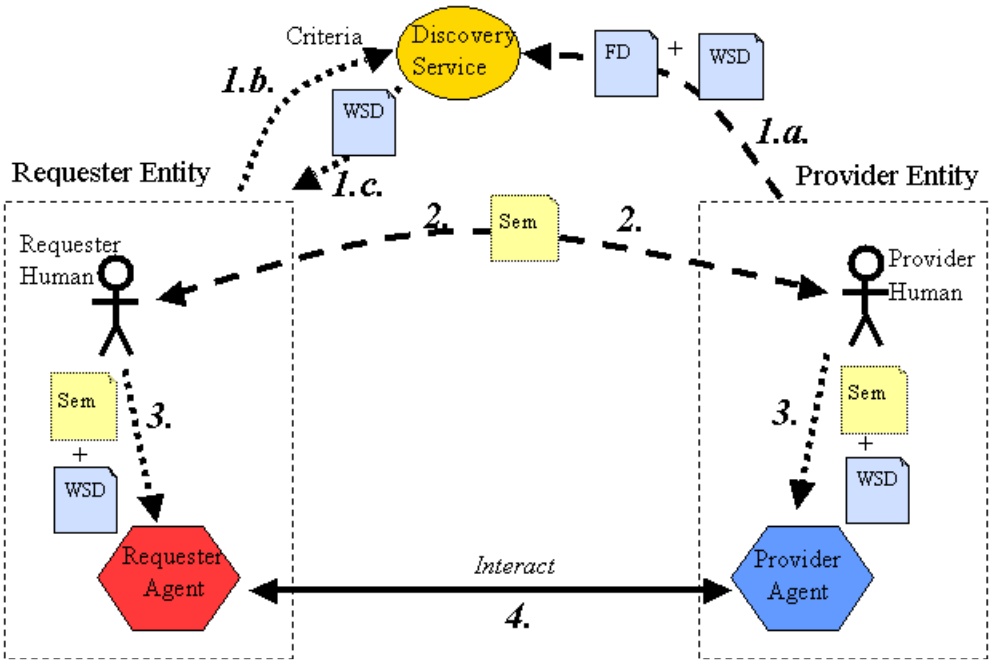
A patto che mittente e ricevente si accordino sulla descrizione del servizio (ad esempio, in un file WSDL), le implementazioni dietro i servizi Web possono essere qualunque.

Scoperta di servizi Web

Per **scoperta** (discovery) si intende la **localizzazione di una descrizione elaborabile in modo automatico di un serivizio Web che risponde a criteri funzionali dati.**

Per **servizio di scoperta** (discovery service) si intende un servizio che aiuta nel processo di scoperta; può essere implementato sia dall'agente provider che dall'agente requester o da un altro agente.

Processo di scoperta:



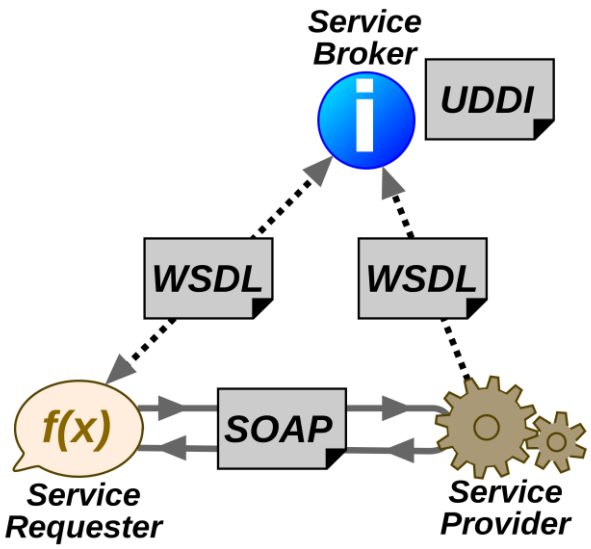
1. Requester e provider divengono noti l'uno all'altro
    a. Il servizio di discovery ottiene sia la descrizione del servizio (WSD = Web Service Description) che la descrizione funzionale associata (FD = Functional Description).
    La complessità della descrizione funzionale è molto variabile: può andare da poche parole di metadati o un URI all'uso di TModel (in UDDI) o una collezione di istruzioni RDF, DAML-S o OWL-S. La descrizione del servizio

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

può essere ottenuta in diversi modi: ad esempio con un search engine, che naviga nel Web raccogliendo tutte le descrizioni di servizi che trova, anche senza l'intervento del provider; oppure con un registro (ad esempio, UDDI), dove si richiede al provider stesso di pubblicare la descrizione e la descrizione funzionale del servizio per il servizio di discovery.

b. I criteri di ricerca specificati dal requester al servizio di discovery per selezionare un servizio Web basato sulla descrizione funzionale ad esso associata, o altre caratteristiche quali il nome del provider o altre caratteristiche del provider.
c. Il servizio di discovery produce una o più descrizioni di servizi che soddisfano i requisiti richiesti. Nel caso in cui siano prodotte più proposte, il requester ne seleziona una.

2. Requester e provider si accordano sulla semantica dell'interazione desiderata: di solito il provider la definisce lasciando al requester l'opzione **prendere** o **lasciare**. Si rende opportuno la definizione di standard. Devono essere d'accordo anche sulla descrizione del servizio, ma questo di fatto è già avvenuto nel passo 1.c.
3. La descrizione del servizio e la sua semantica sono passate agli agenti del requester e del provider.
4. Avviene lo scambio di messaggi SOAP tra i due agenti, da parte dei rispettivi proprietari .



Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Scoperta manuale o autonoma?

- Scoperta **manuale:** l'umano, tipicamente in fase di sviluppo, cerca la descrizione di un servizio che soddisfi i suoi desideri.
- Scoperta **autonoma**: è l'agente del requester che esegue questa ricerca, sia essa eseguita durante lo sviluppo o l'esecuzione.

Ovviamente i due casi richiedono condizioni molto diverse:

- **Requisiti di interfaccia:** in un caso orientati all'umano, nell'altro ad una elaborazione automatica.
- Necessità di **standardizzazione:** più importante per scoperta autonoma;
- **Fiducia:** non è detto che gli umani si fidino di decisioni prese dalle macchine. Per poter arrivare ad una discovery autonoma, è necessaria una semantica che possa venir elaborata automaticamente.

Discovery: registro, indice o peer-to-peer?

Un **registro** (registry) rappresenta un deposito di descrizioni di servizi controllato centralmente e autorevole:

- Viene richiesto al provider un passo preciso: deve collocare l'informazione riguardante il servizio nel registro perché questa possa essere disponibile per gli altri;
- Di conseguenza è il provider a decidere chi ha l'autorità per rendere disponibile l'informazione, che non può quindi essere gestita da terze parti;
- Inoltre, è sempre il provider che decide quale informazione va messa nel registro.

L'**indice** si differenzia nel registro perché pur essendo una raccolta di informazioni esterna non ha né controllo né, di conseguenza, autorevolezza. In questo caso la pubblicazione è passiva: il provider espone sul Web il servizio e la sua descrizione funzionale, e sarà chi prepara gli indici a raccogliere queste informazioni senza che il provider ne venga necessariamente informato.

- Chiunque può preparare un indice, raccogliendo le informazioni usando degli spider e organizzandola in un indice;
- Organizzazioni diverse possono quindi avere indici di questo tipo;
- L'informazione contenuta in un indice può risultare superata: occorre controllarla prima dell'uso. Per questo motivo conterrà riferimenti a informazioni autorevoli che rendono possibile il controllo.
- Un indice può comprendere anche informazioni di terze parti;

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

- Indici diversi possono fornire informazioni di tipo diverso, alcuni più completa, altri meno;
- Saranno le leggi di mercato a decidere quale sarà l'indice effettivamente usato.

La differenza più importante tra registro e indice riguarda chi controlla l'informazione che vi è contenuta.

L'approccio **peer-to-peer** non ha repositories centralizzate: sono i servizi che si scoprono reciprocamente in modo dinamico passandosi le richieste sulla rete.

È più robusto, perché non dipende da un solo punto centrale (o pochi punti centrali). Ogni nodo potrebbe contenere un proprio indice dei servizi Web esistenti. Il tempo di latenza è minore.

Svantaggi tipici delle applicazioni **peer-to-peer**: inefficienza e overhead.

In conclusione:

- I sistemi **P2P** saranno da preferire in ambienti dinamici, in cui vincoli di vicinanza limitano comunque la propagazione delle inondazioni di richieste (ubiquitous computing);
- I **registri** centralizzati si adattano meglio ad ambienti statici e ben controllati, in cui l'informazione non cambia spesso;
- Gli **indici**, infine, sono in grado di adattarsi meglio al crescere delle dimensioni del problema e sostengono la competizione e la differenziazione delle strategie di ricerca.

Discovery e descrizioni funzionali
Condizione necessaria per poter implementare agenti che siano in grado di scoprire servizi è una **descrizione funzionale** del servizio che possa essere proficuamente elaborata in modo automatico.

Occorre standardizzazione, quale OWL-S: OWL per i Servizi (OWL = **Ontology Web Language**: linguaggio basato su logiche descrittive per il Web semantico).

Uso diffuso dei **metadati** come parte della descrizione della semantica del servizio Web leggibile in automatico. L'uso di WSDL permette di specificare la forma dei messaggi attesi, dei tipi diversi elementi dei messaggi e, attraverso l'uso di un linguaggio di descrizione della **coreografia**, il flusso di messaggi attesi tra diversi agent del servizio Web.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

20.2.1 **Parser XML**

Per accedere da un programma ad un documento XML si usa un parser che, a partire da un documento, rende disponibili delle API. Le **API** = Application Programming Interfaces disponibili per un documento XML sono di due tipi:

1. Basate su eventi;
2. Basate su oggetti, a loro volta distinte in:
    a. Basate su oggetti generici;
    b. Basate su oggetti dell'applicazione;

Una prima possibilità è quella di usare la **DOM**, che abbiamo già visto: si tratta di una generica API orientata agli oggetti, standardizzata dal WWW. Essa può essere utilizzata più in generale per leggere, scrivere e trasmettere documenti XML. Un parser basato su DOM restituisce un oggetto di tipo albero corrispondente all'intero documento fornito in ingresso.

Un'altra possibile modalità di elaborazione è quella **basata sugli eventi**: ad ogni passo, il parser rende disponibile all'applicazione il frammento di dati che corrisponde ad un singolo "evento" nel documento di ingresso.

Esempi di eventi sono le etichette di apertura dei vari elementi, le corrispondenti etichette di chiusura, i contenuti testuali.

L'API basata su eventi più diffusa si chiama **SAX** = Simple API for XML.

Si dice che la DOM fornise **oggetti XML generici**, intendendo che la struttura dei suoi dati e i suoi metodi riflettono i costrutti presenti in ogni documento XML: elementi, attributi, dati testuali, etc…

Tuttavia a volte può essere utile avere a disposizione delle strutture dati che siano specifiche dell'applicazione considerata, e in particolare degli oggetti (**oggetti dell'applicazione**) con metodi creati apposta per quella particolare applicazione.

Esistono programmi che offrono supporto nell'operazione di **data binding**, che lega i dati agli oggetti dell'applicazione, e in particolare al **data mapping** da essa richiesto.

Come visto i risultati vengono sempre salvati in una **cache**.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

20.3 **Tecnologie per i servizi Web**

Tre sono le tecnologie fondamentali per l'implementazione di servizi Web:

1. **SOAP** = Simple Object Access Protocol;
2. **WSDL** = Web Services Description Language, per descrivere i servizi disponibili e le modalità per accedervi;
3. **UDDI** = Universal Description Discovery and Integration.

Vediamo ora **come si rende disponibile un servizio Web**
Tre azioni:

1. **Pubblicare:** un servizio Web si fa conoscere;
2. **Trovare:** un'applicazione trova un servizio Web;
3. **Legare** (bind): l'atto di attaccarsi ad un servizio Web e usarne i metodi.

Tre attori:

1. **Fornitore di servizi:** un'applicazione che pubblica e fornisce un servizio su una rete;
2. **Richiedente:** un'applicazione che cerca un servizio per usarlo;
3. **Intermediario:** per far incontrare fornitori e richiedenti attraverso un registro di servizi.

- Per pubblicare un servizio Web, il fornitore aggiunge la corrispondente descrizione al registro;
- Per trovare un servizio, il richiedente descrive il servizio e l'intermediario restituisce informazioni su tutti i servizi Web che corrispondono alla richiesta.
- Dopo che il richiedente ha selezionato un particolare servizio, il richiedente contatta il fornitore e dà inizio alla transazione

21) Architettura ai micro-servizi

Il concetto di servizio viene portato alle estreme conseguenze nelle architetture basate su micro-servizi (**microservice architecture**), che oggi vengono seguite da un considerevole numero di sistemi. L'idea è di costruire il sistema combinando un numero di servizi autonomi che svolgono funzioni elementari.

Ovviamente tali micro servizi devono essere accoppiati in modo lasco (**loosely coupled**) e devono comunicare l'uno con l'altro scambiandosi messaggi basati sulle loro API. Molto spesso questa comunicazione avviene via HTTP.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Anche una **Service-Oriented Architecture** (SOA) è caratterizzata da grande modularità e da comunicazioni basate su messaggi tra i diversi moduli.

Nel caso dei micro-servizi ogni servizio è associato ad una funzione elementare, anche se non esiste un criterio universalmente riconosciuto per decidere quando la funzione è sufficientemente elementare e il servizio può di conseguenza essere definito un micro-servizio.

I sistemi web tradizionali (cosiddetti monolitici o monolitich in inglese), erano caratterizzati da una architettura a tre strati (three-tier architecture): **user-agent, server web** e **information source**.

Coi servizi web siamo passati ad una struttura multi-strato (multi-tier architecture), in cui al posto dello strato di information source posso avere un altro server web e via così.

Il maggior problema con questo tipo di sistemi è che ogni cambiamento, anche se piccolo, richiede di ricostruire (**rebuilding**) e rilasciare (**deployment**) l'intero sistema.

21.1 **Vantaggi di una microservice architecture**

Organizzazioni diverse possono raggiungere vantaggi diversi.

- Ridurre le dipendenze tra le diverse squadre di lavoro, col risultato di velocizzare la produzione di codice;
- Permettere lo svolgimento in parallelo di iniziative diverse;
- Offrire supporto a una plurità di tecnologie, di linguaggi, di piattaforme;
- Offrire aiuto all'innovazione visto che è facile fare a meno di una parte del codice nel caso in cui l'innovazione proposta fallisca.

Quindi da una parte viene potenziata la velocità di produzione del software, ma dall'altra anche la sua sicurezza, legata alla facilità di fare il testing e il deploy individualmente di ogni singolo servizio.

21.2 **Cosa serve per costruire un'architettura ai micro-servizi**

Per costruire un'architettua distribuita che davvero riesca a rendere ogni micro-servizio completamente indipendente da ogni altro occorre intervenire a molti livelli.

Faremo riferimento a due delle tematiche più importanti, ovvero la **progettazione delle interfacce** e la **persistenza dei dati.**

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

21.2.1 Le API

Risulta chiaro che ogni micro-servizio deve essere il più possibile autonomo e slegato dagli altri o da un contesto.

Risulta chiaro che per poter essere utile all'aplicazione è essenziale come comunica con gli altri micro-servizi.

Ogni micro-servizio è dotato di un'interfaccia o **API**: se vogliamo mantenere un accoppiamento lasco tra i diversi microservizi, sarà necessario disaccoppiare il più possibile anche le loro API. Solo in questo modo ogni servizio potrà venir rilasciato (**deployed**) indipendentemente da tutti gli altri e, questa è una proprietà cruciale per ottenere vantaggi di una architettura ai microservizi.

Le API dei micro-servizi sono tipicamente o orientate ai messaggi o in style **hypermedia.**

API orientate ai messaggi
I messaggi scambiati permettono sia di esporre degli ingressi ad un componente, sia di scambiarsi delle informazioni che dipendono dal singolo task. Questo permette di poter fattorizzare i cambiamenti legati all'evoluzione dell'applicazione attraverso variazioni dei messaggi scambiati. Ovviamente ogni azienda fa scelte diverse su quali formati e quali protocolli utilizzare per scambiare messaggi tra micro-servizi.

API in stile hypermedia
Questo secondo tipo di comunicazione è in realtà un arricchimento del primo. Quello che viene scambiato non sono più semplicemente informazioni, ma un mix di informazioni possibili e azioni simile a quello che possiamo fare con HTML, dove l'ipertesto contiene anche form e link che chi riceve può usare per compiere delle azioni. Bisogna infatti pensare che chi progetta le API dei micro-servizi sono in genere esperti di sistemi web.

21.2.2 La persistenza dei dati

Un punto centrale nelle applicazioni è la progettazione dello schema delle basi di dati, attorno a cui poi si muovono le operazioni necessarie per portare a termine le operazioni necessarie alle applicazioni. Ma questo risulta essere un grave problema in un sistema distribuito come quello formato da micro-servizi. Infatti se due servizi devono fare accesso (in lettura e/o scrittura) ad una stessa base di dati, se non addirittura alla stessa tabella nella base di dati, essi risultano ovviamente fortemente accoppiati, ed è impossibile farne il deploying indipendentemente.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

La progettazione non deve più partire dai dati, ma dalle azioni. L'ideale sarebbe che ogni micro-servizio potesse avere un proprio micro-database a cui accedere. Quando questo è possibile, rappresenta una soluzione. Purtroppo questo non è sempre percorribile.

Ci sono sostanzialmente due soluzioni per affrontare questo tipo di situazioni. Il primo viene chiamato **event sourcing** e richiede di salvare i singoli passi anziché il risultato finale, ovvero lo stato dell'oggetto che consideriamo. Ovviamente dalla lista dei singoli passi io posso ricavare il risultato finale, e quindi non perdo informazioni, anzi, semmai ne conservo di ridondante.

Una strategia alternativa per affrontare questo problema si basa su **Command Query Responsability Segregation** (CQRS) e in particolare su una completa divisione tra azioni che cambiano un'entità rispetto a quelle che semplicemente la leggono.

Laddove sia possibile conviene evitare event sourcing e CQRS che finiscono sempre per complicare l'applicazione e vanno quindi implementate solo quando strettamente necessario.

22) Semantic web e Linked Open Data

22.2 **Semantic Web**

Nella prima parte del corso abbiamo visto come il web possa essere visto come un gigantesco ipertesto distribuito sui diversi server HTTP. Tale ipertesto può venir rappresentato come un grafo in cui i nodi corrispondono a documenti, tipicamente a documenti HTML o XML, ma anche in altri formati, e sono collegati l'uno all'altro tramite link che vanno da un determinato punto di un documento ad un punto o dello stesso documento o di un documento diverso.

Quando dal web si passa al **semantic web**, questo ipergrafo diventa una struttura che può venir navigata automaticamente per ricercare e combinare informazioni annotate semanticamente secondo uno standard che può venir elaborato automaticamente.

Trattando di XML, abbiamo discusso di come venga utilizzato per annotare non solo dei documenti, ma anche dei dati. Diviene quindi interessante rendere raggiungibili via web una serie di dati che possono tornare utili agli utenti, annotandoli in modo appropriato e collegandoli tra loro per rappresentare le relazioni che li legano.

Si tratta di pensare al web come ad una gigantesca base di dati che ha però una particolare struttura: non più relazionale, né ad albero, ma a **grafo**. Infatti il web

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

contiene una grande quantità di dati che però così come sono, sono difficilmente utilizzabili dalle applicazioni, perché in formati come l'HTML.

Le azioni che si prefigge il semantic web in questo campo sono quindi diverse:

- Rendere i dati sul web fruibili da applicazioni di intelligenza artificiale che possano elaborarli in maniera intelligente e fornire quindi delle informazioni più vicine a quello che serve all'utente finale.
- Incoraggiare aziende, organizzazioni e singoli a pubblicare i loro dati in modo gratuito, seguendo standard aperti;
- Incoraggiare le aziende ad usare i dati già disponibili sul web.

L'introduzione dei **linked data** permette di vedere il web come una sorta di enorme base di dati, in forma di grafo, in cui ogni nodo corrispondente ad una **risorsa**, ogni arco una **proprietà** di questa risorsa che la lega ad un nodo che rappresenta il **valore della proprietà**.

Linked Open Data o LOD che hanno l'ulteriore caratteristica di essere aperti, ovverro accessibili senza restrizioni.

22.3 **Linked data, open data e linked open data**

Dei dati possono venir messi a disposizione in modo che possano venir liberamente utilizzati, riciclati e redistribuiti senza alcuna restrizione, se non, in alcuni casi, che i risultati di tali operazioni vengano resi anch'essi fluibili alle medesime condizioni. In questo caso si parla di dati operati o **open**.

Il concetto di **Linked Data** è invece più tecnico e si riferisce all'interoperabilità tra dati provenienti da sorgenti diverse.

La differenza fondamentale tra documenti e dati è che questi ultimi hanno più struttura rispetto ad un documento, ed in particolare ad un ipertesto. Si parla di Linked Data per riferirsi a come dati strutturati posono venir pubblicati e messi in relazione tra loro sul web.

Ovviamente un'importante differenza è che mentre i documenti sono indirizzati all'utente umano, i dati strutturati sono tipicamente pensati per un accesso automatico all'informazione.

Dare quindi a questo tipo di dati una forma che ne faciliti l'interoperabilità rappresenta la chiave per favorire la lettura e l'utilizzo, arrivando non solo alla cercata trasparenza, ma anche alla possibilità di trovare collegamenti tra dati che provengono da sorgenti diverse.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Mentre il concetto di **open** si riferisce ad una politica di diffusione dei dati, e quindi di licenza d'uso, la possibilità di avere dati tra loro connessi, soprattutto in presenza di grandi quantità di dati, si poggia su considerazioni tecniche e su metodologie tipiche del mondo web.

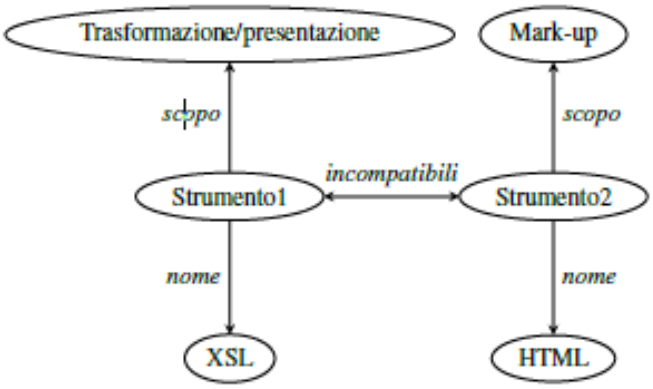
Esistono diversi modelli possibili per rappresentare ed implementare la relazione tra due entità, ma quella che è risultata vincente in questo campo, trovando un buon compromesso tra possibilità di rappresentare l'informazione e efficienza nell'elaborazione, si basa sul **Resource Description Framework** o **RDF**.

22.4 **Rappresentazione dei dati sotto forma di grafo**

Una base di dati **relazionale** è composta da un certo numero di tabelle legate tra loro attraverso le chiavi primarie. In una base di dati **gerarchica**, quali ad esempio quelle basate su XML, abbiamo una serie di nodi, che per l'XML corrispondono agli elementi, che sono legati tra loro da relazioni (padre,figlio). Per cui il padre è più importante del figlio. Nel caso di XML il figlio è sottoelemento rispetto al padre.

In una base di dati **grafica** i nodi sono legati tra loro da relazioni orientate che però non presuppongono alcuna relazione di maggiore o minore importanza tra i due nodi coinvolti.

Ogni nodo corrisponde ad una **risorsa**, che può essere legata ad altre risorse, ma senza che questo implichi che una risorsa è più importante delle altre.



22.5 **RDF – Resource Description Framework**

L'acronimo RDF sta per Resource Description Framework, ovvero framework per la descrizione di risorse, si fa capire come un concetto fondamentale al suo interno sia proprio quello di **risorsa**.

Esso non ha in sé alcuno strumento per associare una semantica ai dati.

Tali strumenti comprendono **dizionari** e **ontologie**, con i propri strumenti di rappresentazione.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Lo scopo di RDF è quello di rappresentare l'informazione in modo che sia facile da leggere ed elaborare via calcolatore, non per rappresentarla all'utente umano.

RDF può essere rappresentato tramite XML, ottenendo RDF/XML. È facile capire RDF se si parte dal modello grafico che abbiamo introdotto precedentemente.

Esempio di documento RDF/XML:

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">

<rdf:Description
rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
  <cd:artist rdf:resource="http://www.somesite.com/Bob Dylan"/>
  <cd:country>USA</cd:country>
  <cd:company>Columbia</cd:company>
  <cd:price>10.90</cd:price>
  <cd:year>1985</cd:year>
</rdf:Description>

<rdf:Description
rdf:about="http://www.recshop.fake/cd/Hide your heart">
  <cd:artist rdf:resource="http://www.someothersite.com/Bonnie Tyler"/>
  <cd:country>UK</cd:country>
  <cd:company>CBS Records</cd:company>
  <cd:price>9.90</cd:price>
  <cd:year>1988</cd:year>
</rdf:Description>
</rdf:RDF>
```

La prima linea contiene la dichiarazione di documento XML, seguita dall'elemento radice che ha etichetta **<rdf:RDF>**. Vengono definiti due namespaces: **rdf**, che riguarda caratteristiche appunto dell'rdf, e **cd**, che si adatta al fatto che ci si sta occupando di CD. Il namespace **rdf** è ovviamente essenziale alla definizione di un documento RDF.

L'elemento **<rdf:Description>** contine la descrizione della risorsa identificata dall'attributo **rdf:about** ed è alla base della struttura dell'RDF.

L'URI identifica univocamente una risorsa: l'attributo **rdf:about** contiene quindi un **identificativo** della risorsa di cui stiamo trattando.

Gli elementi con etichetta **<rdf:Description>**, comprese le etichette aperta e chiusa, vengono quindi detti **Statements RDF**.

I sottoelementi di **<rdf:Description>** corrispondono a **predicati** aventi per soggetto la risorsa: **<cd:artista>**, **<cd:nazione>**, **<cd:casa>**, etc.

Giordano Vincenzo N86003039
vincenzo.giordano14@studenti.unina.it

Si noti come gli **oggetti** dei predicati possono essere dati dal contenuto dell'elemento o da un'altra risorsa, indicata dall'attributo **rdf:resource.**

Questo ovviamente corrisponde ad un grafo in cui la proprietà, corrispondente ad un arco, può puntare ad una foglia (valore) o ad un nodo interno (risorsa).

Un elemento **rdf:Description** ha quindi la seguente struttura:

```
<rdf:Description rdf:about="subject">
  <predicate rdf:resource="object">
  <predicate> literal value </predicate>
<rdf:Description>
```

e corrisponde a **due** statements RDF che si riferiscono alla stessa risorsa, il subject.

Nel primo caso l'oggetto è dato da una risorsa, nel secondo da un letterale.