

# CGI

Ing Cesare Monti  
(revisionato da Andrea Omicini)

## HTTP: CGI



- Nate per aggiungere un modello di interazione tra client e server
  - la computazione sul server può essere specializzata
- Il codice risiede interamente sulla macchina server
  - ... questo spiega il termine server side ...
- Si lascia aperta la possibilità di eseguire codice chiamandolo dal web
- Questa possibilità viene per la prima volta consentito dal protocollo CGI
  - Common Gateway Interface

## CGI: il protocollo

- I clients richiedono l'esecuzione di un programma
  - [oggi non sembra nulla... ma allora era un passo avanti notevole]
- I server invocano il programma chiamato nell'URL
- e utilizzano il protocollo CGI per interpretare il metodo (GET, POST) con cui passare i parametri al programma invocato (via stdin)
- Il programma viene eseguito
- Una volta eseguito, ritorna la risposta in formato HTML (via stdout) al server Web
  - [o meglio il programmatore ritorna in HTML via stout, e c'è ridirezione (più o meno) automatica]
- Il server Web rigira la risposta al client

## CGI: altre migliorie – lo stato

- Tramite CGI è stato introdotto il concetto di stato legato all'HTTP
- Lo stato viene mantenuto attraverso l'invio di variabili che vengono mantenute in memoria dal client (cookies)
  - ... il nome pare sia stato coniato da un programmatore Netscape

<http://www.cookiecentral.com/faq/#1.2>

## CGI: how to

- per aggiungere passaggio di parametri tra una connessione e l'altra anche HTML è stato modificato
  - aggiunta di
    - Form
      - `<form action="..." method="...">`
    - Form Elements
      - `<input type="...">`
  - <http://www.w3c.org>

## CGI: How To

- Possono essere scritte in qualsiasi linguaggio che possa venire interpretato dalla macchina server
- Quelli più usati sono
  - C/C++
  - Fortran
  - PERL
  - Python
  - TCL
  - any Unix shell language
  - Visual Basic
  - AppleScript
  - Java

## CGI: passaggio dei dati

- GET
  - viene riscritto a run time l'URL della risorsa a cui si vuole accedere aggiungendo i campi che si vogliono passare all'URL stesso
- Es:
 

```
<chiamata a: http://www.unSito.com/unoScriptCgi.exe>
```

 con parametri: param1="10", param2="ciao"
- La request line diventa
  - `GET www.unSito.com/unoScriptCgi.exe?param1="10"&param2="ciao" HTTP/1.0`

## CGI: il passaggio dei dati

- POST
  - Tutti i parametri vengono passati dentro al campo Entity Body e viene modificato il method del Request Line
- Es:
 

```
<chiamata a: http://www.unSito.com/unoScriptCgi.exe>
```

 con parametri: param1="10", param2="ciao"
- `POST www.unSito.com/unoScriptCgi.exe HTTP/1.0`

```
...
param1="10"
&param2="ciao"
```

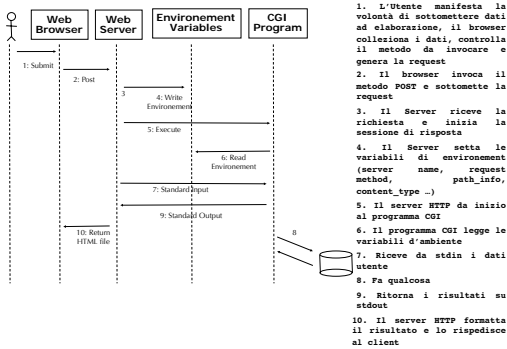
## CGI: pro e contro dei metodi

- GET
  - Pro
    - i parametri rimangono visibili all'utente
    - l'output della pagina dinamica può essere bookmarkato
  - Contro
    - si deve essere sicuri che lo script CGI reso disponibile non possa eseguire azioni dannose a fronte di parametri sbagliati
      - hacking
    - nella stessa dello script occorre fare parsing sull'URL per avere i parametri

## CGI: pro e contro dei metodi

- POST
  - Pro
    - non occorre dividere URL dai parametri
      - è già nel metodo
  - Contro
    - nel caso di pacchetti incompleti non si può eseguire nulla
    - le pagine non possono essere "bookmarkate"

## CGI: scenario



## Example: C code

```

main(int argc, char *argv[]) {
    entry entries[MAX_ENTRIES];
    register int x,m=0;
    int c1;
    printf("Content-type: text/html%c%c",10,10);
    // CHECK SUL CONTENT TYPE
    if(strcmp(getenv("CONTENT_TYPE"),"application/x-www-form-urlencoded") {
        printf("This script can only be used to decode form results. \n");
        exit(1);
    }
    c1 = atoi(getenv("CONTENT_LENGTH"));
    // ACQUISIZIONE DATI DA STDIN
    for(x=0;c1 && (!feof(stdin));x++) {
        m=x;
        entries[x].val = fkeyword(stdin, '&', &c1);
        entries[x].name = makeword(entries[x].val, '=');
    }
    // PREPARAZIONE DATI DI OUTPUT
    printf("<html>Query Results</html>");
    printf("You submitted the following name/value pairs:<p>");
    printf("<ul>");
    for(x=0; x <= m; x++)
        printf("<li> <code>%s = %s</code>",&entries[x].name,
            entries[x].val,10);
    printf("</ul>");
}
    
```

## CGI: dove sta la fregatura?

- Il protocollo CGI prevede l'istituzione di un nuovo processo ogni qual volta si invochi una CGI
- pensate quindi che ad ogni request parte un processo
  - pensate ad un server web con molta utenza ...
- ...col tempo è stata introdotto il protocollo FastCGI ... ma non sempre è applicabile
  - non molto diffuso

## CGI ... e dopo ?

- scrivere CGI implica scrivere un applicativo che produca il proprio output codificato in HTML
- implica anche una serie impressionante di problemi di gestione
  - legati alla scalabilità dell'ambiente
  - e alla eterogeneità dei client
    - quanto applicativi browser esistono??

## CGI ... e dopo ?

- col tempo l'evoluzione delle CGI ha portato ad una serie di:
  - linguaggi di elaborazione server-side
    - PHP
    - JSP
    - ASP
  - Ognuno dei quali ha propri meccanismi per abilitare e gestire l'interazione e la comunicazione
    - beans
    - sessioni
    - oggetti server

## CGI ... e dopo ?

- evoluzione degli applicativi server
  - da web server a Application server
    - JBoss
    - Cocoon
    - ...
  - ognuno con proprie specifiche e capacità

## CGI

- un passo per volta
- qualche link
  - <http://www.cgi101.com/book>
  - <http://hjs.geol.uib.no/Cplusplus>