CGI

Ing Cesare Monti - 20 aprile 2005

HTTP: CGI

- Nate per aggiungere interazione tra client e server
- 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 aderisce ad un protocollo di Common Gateway Interface

CGI : il protocollo

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

CGI : Ulteriori 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 derivi 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
      - http://www.w3c.org/Consortium/membership

CGI: How To

- Possono essere scritte in qualsiasi linguaggio che possa venire interpretato dalla macchina server, quelli più usati sono:
  - **C/C++**
  - **Fortran**
  - **PERL**
  - **Phyton**
  - **TCL**
  - **Any Unix shell**
  - **Visual Basic**
  - **AppleScript**
  - **Java**

CGI: il passaggio dei dati

- GET
  - viene riscritto 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"
- Il 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”

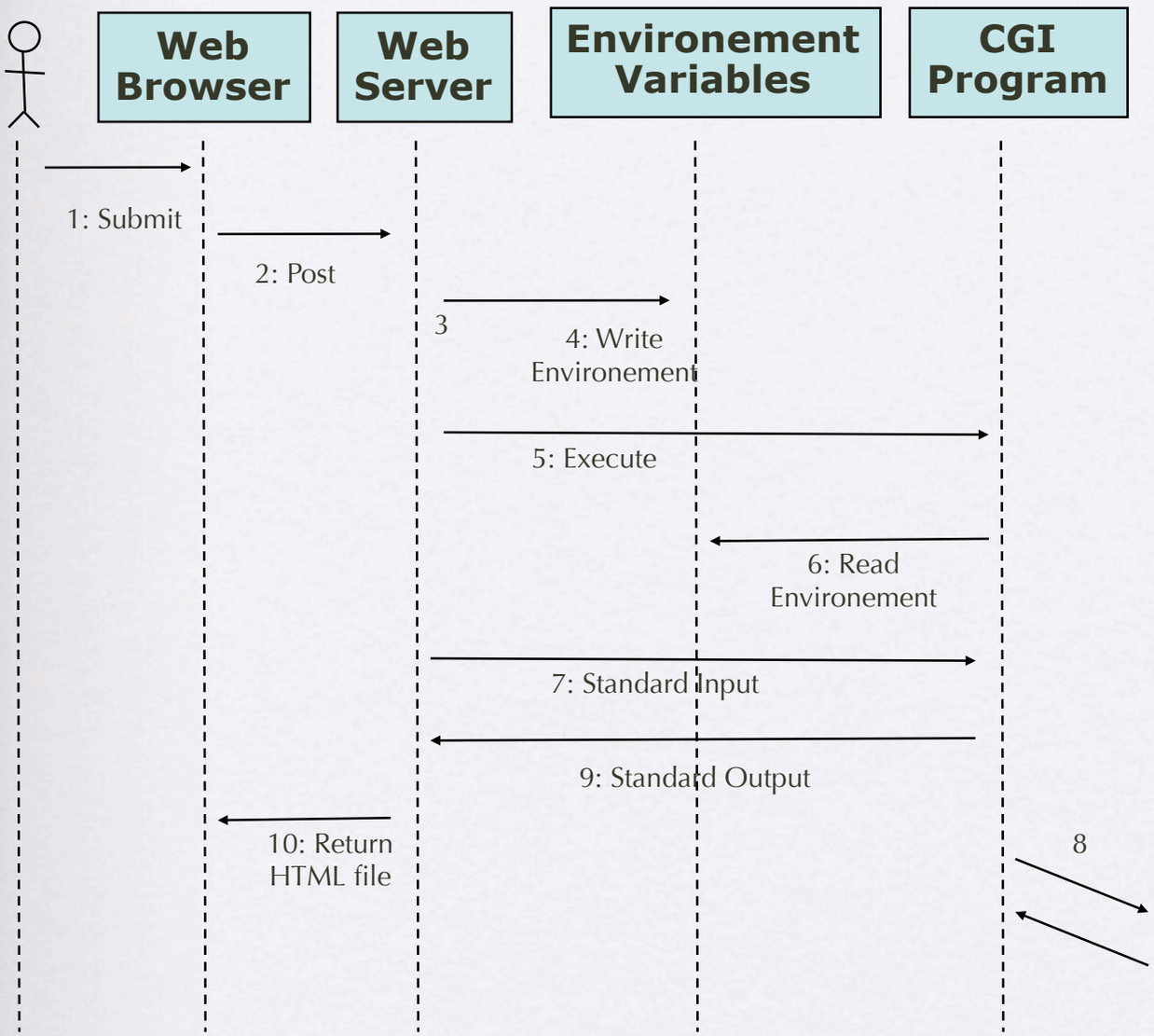
CGI: pro e contro dei metodi

- GET
  - Pro:
    - i parametri rimangono visibili all'utente
    - l'output della pagina dinamica può essere bookmarcato
  - Contro:
    - si deve essere sicuri che lo script CGI reso disponibile non possa eseguire azioni dannose a fronte di parametri sbagliati
    - nella stesura 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
  - Contro:
    - nel caso di pacchetti incompleti non si può eseguire nulla
    - le pagine non possono essere bookmarcate

CGI: scenario



1. L'Utente manifesta la volontà di sottomettere dati ad elaborazione, il browser colleziona i dati, controlla il metodo da invocare e genera la request
2. Il browser invoca il metodo POST e sottomette la request
3. Il Server riceve la richiesta e inizia la sessione di risposta
4. Il Server setta le variabili di environement (server name, request method, path_info, content_type …)
5. Il server HTTP da inizio al programma CGI
6. Il programma CGI legge le variabili d'ambiente
7. Riceve da stdin i dati utente
8. Fa qualcosa
9. Ritorna i risultati su stdout
10. Il server HTTP formatta il risultato e lo rispedisce al client

Example: C code

```
main(int argc, char *argv[]) {
    entry entries[MAX_ENTRIES]];
    register int x,m=0;
    int cl;
    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);
    }
    cl = atoi(getenv("CONTENT_LENGTH"));
// AQUISIZIONE DATI DA STDIN
    for(x=0;cl && (!feof(stdin));x++) {
        m=x;
        entries[x].val = fmakeword(stdin,'&',&cl);
entries[x].name = makeword(entries[x].val,'=');
    }
// PREPARAZIONE DATI DI OUTPUT
    printf("<H1>Query Results</H1>");
    printf("You submitted the following name/value pairs:<p>%c",10);
    printf("<ul>%c",10);

    for(x=0; x <= m; x++)
        printf("<li> <code>%s = %s</code>%c",entries[x].name,
                entries[x].val,10);
    printf("</ul>%c",10);
}
```

CGI: dove sta la fregatura?

- Il protocollo CGI prevede l'istanziamento 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

CGI ... e dopo ?

- scrivere CGI implica scrivedre un applicativo che produca il proprio output codificato in HTML
- implica anche una serie impressionante di problemi di gestione
  - legati alla scalabilità dell'ambiente
  - ed 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

CGI ... e dopo ?

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

CGI

- un passo per volta
- qualche link
  - http://www.cgi101.com/book
  - http://cgipoint.html.it/tutorial
  - http://hjs.geol.uib.no/CplusPlus