Capitolo 3

SOFTWARE E LINGUAGGI DI PROGRAMMAZIONE UTILIZZATI

In questo capitolo vedremo i programmi ed i linguaggi utilizzati per implementare i concetti esposti nel primo capitolo, evidenziandone le caratteristiche ed accennando, ove necessario, la sintassi delle varie istruzioni. Per approfondimenti ulteriori invece si rimanda ai manuali di riferimento disponibili on-line. Ricordiamo che uno degli scopi del presente lavoro è quello di rendere disponibile su **Web** i risultati ottenuti, sia per la consultazione dei procedimenti utilizzati, sia come elaborazione in tempo reale delle parole immesse nel sistema. Come visto nel capitolo precedente, il sintetizzatore utilizzato per questo lavoro richiede una piattaforma **Windows** compatibile per poter girare. Poiché non è automatizzabile l'apertura del programma da un altro applicativo, è comunque possibile far girare il programma su di un'altra piattaforma ed importare solo il file da sintetizzare. Per comodità operativa nel nostro caso si è utilizzato **Windows 2000**. I criteri guida nella scelta degli applicativi sono stati molteplici:

1) software in grado di integrarsi in maniera semplice con le pagine **Web**

2) grande flessibilità nella gestione delle stringhe

3) realizzazione di un programma multipiattaforma il cui funzionamento fosse indipendente dal sistema operativo

4) contenimento della spesa per l'uso dei software necessari

Il primo punto riguarda la possibilità di realizzare un sistema user frendly, basato sulle note e largamente diffuse pagine **Web**, permettendo l'accesso tramite rete **Internet** a tutto il lavoro sviluppato.

l secondo riguarda la presenza nel linguaggio di programmazione di funzioni potenti e versatili nella elaborazione delle stringhe, che non si limitino alla semplice concatenazione o estrazione di sottostringhe.

Il terzo assicura l'adattabilità del progetto a qualsiasi tipo di sistema operativo.

L'ultimo criterio viene soddisfatto utilizzando software di tipo freeware, rendendo assolutamente svincolato il progetto da eventuali obblighi economici per l'acquisizione di licenze e quindi a spesa minima, al di là del costo marginale del lavoro di programmazione.

Ottemperare a tutte queste esigenze è stato possibile tramite l'impiego di un pacchetto di programmi

gratuito che va sotto l'acronimo di **L.A.M.P**. Ogni lettera è l'iniziale del nome di un software: **L** sta per **Linux**, il sistema operativo che in questo caso non è stato utilizzato per quanto detto sul sintetizzatore **HLsyn**; **A** sta per **Apache**, ed è un **Web** server che copre il 67% del mercato ed è completamente gratuito, **M** sta per **Mysql**, ed è un database; **P** è l'iniziale di **PHP**, un linguaggio di programmazione estremamente versatile che richiama molte strutture del **PERL** e del **C++.** Oltre questi software, sono stati utilizzati il programma **Spyguru**, per la decodifica del file di testo necessario per caricare i dati all'interno del sintetizzatore; il **JavaScript**, per operazioni di verifica ed elaborazione lato client, il **DISC 95**, un vocabolario in formato elettronico che ha permesso di rilevare proprietà statistiche della lingua italiana e ricavare un nutrito database di parole utilizzate per il corretto funzionamento dell'intero progetto. Per finire citiamo il programma di scrittura **Word 2000** con il quale è stata redatta la tesi.

3.1 PERCHE' PHP?

Oltre i motivi sopra riportati ci sono altri aspetti che hanno fortemente influito nell'adozione di questo linguaggio di programmazione: ogni pagina **Web** è scritta con un linguaggio denominato **HTML**, che permette di gestire la formattazione di una pagina con l'inserimento di testo, di links, immagini, sfondi etc.. in maniera rigida e determinata. Infatti, potremmo dire che l'**HTML** non rappresenta un vero e proprio linguaggio di programmazione, ma più una sorta di specifiche relative ad i vari attributi, in quanto è possibile con qualsiasi editor di pagine **Web**, posizionare gli elementi desiderati nella pagina ed ottenere il codice relativo, cosa impossibile per qualsiasi linguaggio di programmazione vero e proprio: la categoria dei programmatori inciderebbe in maniera drammatica sul tasso di disoccupazione mondiale! Si intuisce come una siffatta pagina **Web** non possa interagire con un eventuale utente o comunque con un sistema remoto, in quanto rappresenta una semplice visualizzazione di più elementi assemblati insieme: è, cioè, una pagina **statica**.

Per far si che la pagina **Web** diventi dinamica e quindi attiva, occorre poter prelevare i dati inseriti, analizzarli ed inviarli ad un'altra pagina deputata all'elaborazione tramite un protocollo specifico e riceverne il risultato.

Esistono due tipi di programmi nel mondo **Web**: uno detto lato **client**, cioè residenti sull'**host** richiedente il servizio, come ad esempio il **Javascript**, ed uno detto lato **server**, cioè residenti sull'**host** server. Per questi ultimi, la connettività può essere realizzata con due modalità: tramite l'uso di **CGI** (common gateway interface), cioè interfacce appositamente studiate per dialogare con il **Web server** ed usate ad esempio nel **Perl**, o tramite l'uso di linguaggi "immersi" nel codice **HTML** come l'**ASP**.

Il **PHP** fa parte di questa famiglia e può essere tranquillamente incorporato nelle varie parti di codice **HTML**. Il **PHP** è un linguaggio interpretato ed il suo output rappresenta proprio codice **HTML**. La sua sintassi deriva direttamente dal **C/C++** e richiede un brevissimo periodo di addestramento per chi ha già esperienza di programmazione con il suddetto linguaggio. Essendo poi un codice di tipo immerso, la generazione di codice **HTML** risulta semplificata in confronto all'uso di **CGI**. L'aspetto fondamentale del suo uso in questo contesto, risiede nella vastissima libreria di funzioni dedicate alle stringhe, estremamente versatile, ed alla implementazione di una tecnica di programmazione che va sotto il nome di "espressioni regolari" o **REGEXP**. Inoltre, presenta piena compatibilità con le **PCRE**, le espressioni regolari del **Perl**, forse il linguaggio più potente per l'elaborazione delle stringhe.

Come abbiamo visto nel capitolo I, dovremo fare un numero elevato di verifiche su parole e contesti fonologici e, quindi, controlli di elevata complessità su combinazioni di caratteri e stringhe per

riscontrare la presenza o meno di determinate proprietà. Queste operazioni risultano particolarmente semplici ed immediate con l'uso delle espressioni regolari permettendo di realizzare un codice compatto ed elegante a vantaggio sia della leggibilità che delle prestazioni. Descriviamo adesso le istruzioni, le funzioni e le espressioni regolari utilizzate per la stesura dei programmi, descrivendone le caratteristiche e la sintassi.

Nel **PHP** a differenza del **C**, non è necessario dichiarare le variabili prima del loro utilizzo ma basta semplicemente scriverne il nome per renderla disponibile alla parte di codice seguente. Al nome che contrassegna la variabile deve essere anteposto il simbolo **"$"**. Analogamente agli altri linguaggi il simbolo **"="**, assegna il valore riportato alla sua destra alla variabile scritta alla sua sinistra. Se il valore da assegnare ad una variabile è racchiuso fra virgolette, viene interpretato come una sequenza di caratteri o più propriamente una stringa. Facciamo un esempio:

$parola = "gioco";

con questa istruzione si è creata una variabile di nome parola alla quale viene assegnato un valore di tipo stringa – *gioco* -. Il simbolo di **";"** indica la fine dell'istruzione. Si noti inoltre come non sia necessario specificare il tipo di variabili, che risulta implicito nell'operazione di assegnazione. Questo aspetto, benché comodo, potrebbe portare difficoltà di lettura o addirittura ad errori di programmazione nel caso si dichiarino operazioni di conversione di tipo (**casting**): occorre fare attenzione ad utilizzare nomi diversi per le variabili oppure tenere memoria dell'ultima conversione affinché possano essere correttamente svolte le operazioni successive.

I tipi di dato presente nel **PHP** sono quelli che comunemente troviamo in qualsiasi linguaggio: interi, reali, stringhe, vettori, vettori associativi. Ricordiamo che un vettore associativo è un vettore che invece di utilizzare un indice numerico per la sua scansione, utilizza delle stringhe denominate parole chiave.

Le operazioni di tipo aritmetico, logico, per l'esecuzione di cicli (for o while) o verifiche di condizione (if else) sono quelle implementate in qualsiasi linguaggio di programmazione e la sintassi è simile a quella del linguaggio **C**. In questo linguaggio non è obbligatoria l'indentazione, benché utilizzata in questo lavoro per rendere più leggibile e chiaro il programma. Saranno invece esaminate con un certo dettaglio le operazioni e le funzioni applicabili alle stringhe, corredandole con esempi per capirne sia la funzionalità che la sintassi; le operazioni utilizzabili sui vettori; le istruzioni per prelevare i dati dal protocollo **HTTP** o per renderle disponibili in una sessione di lavoro, le righe di codice necessarie per potere interfacciare il **PHP** con il database **Mysql**, le funzioni per la scrittura e la lettura da file.

3.1.1 Operazioni su stringhe

$parola=**"gioco";**

La variabile indicata fra virgolette è considerata di tipo stringa.

$lunghezza=**strlen**($appoggio);

La funzione **strlen** permette di misurare la lunghezza della stringa contenuta fra parentesi. Il valore viene assegnato alla variabile $fine.

$stringa=**substr**($appoggio,0,$pos)**.substr**($appoggio,$pos+1,$fine);

In questa espressione troviamo l'operatore di concatenazione "**.**" Usato per unire variabili stringhe e l'operatore **substr,** che genera ancora variabili di tipo stringa. Gli argomenti rappresentano nell'ordine: la stringa di cui si vuole prendere una parte, il punto iniziale della sottostringa scelta e la sua lunghezza. Questi valori possono essere numeri o anche variabili come nel nostro esempio.

3.1.2 Operazioni su vettori

$vocali=**array**("à"=>"a","è"=>"e","ò"=>"o","é"=>"e","ó"=>"o","ì"=>"i","ù"=>"u");

Questa espressione permette di creare un array inserendone i valori in un sol colpo. E' anche possibile aggiungerne in sequenza con l'espressione:

**$vocali[]=**$valoredaimmettere;

In questo esempio si è creato un array associativo, utilizzando come campi chiave le lettere indicate fra virgolette. Omettendo i campi chiave si sarebbe ottenuto un vettore con la classica indicizzazione.

$vettore=**explode**("-",$stringa);

Questa espressione permette di ottenere un vettore da una stringa utilizzando come simbolo separatore quello contenuto fra virgolette. Ad esempio la parola *ca-sa,* verrebbe memorizzata dopo l'**explode,** in un array bidimensionale il cui primo elemento conterrebbe *ca*, ed il secondo *sa*.

$stringa=**implode**("",$vettore);

Questa espressione attua esattamente la funzione inversa alla precedente. Anche in questo caso si può utilizzare un simbolo di unione. Si veda anche la funzione **join**.

count($vettore);

Questa funzione restituisce il numero di elementi contenuti in un vettore.

array_pop($vettore);

Questa funzione elimina l'ultimo elemento di un vettore.

array_push($vettore,$ultimoelemento);

Questa funzione inserisce come ultimo elemento di un vettore la variabile contenuta nel secondo argomento.

$vettore3=**array_merge**($vettore1,$vettore2);

Questa funzione restituisce un terzo vettore contenente gli elementi dei vettori passati come argomento.

3.1.3 I dati nel protocollo HTTP

$stringaapp=**$HTTP_GET_VARS**[caratteri];

Il valore nell'array associativo contenente i dati trasferiti su protocollo HTTP in modalità GET, è caricato nella variabile $stringaapp.

3.1.4 Operazioni su file e funzioni

$file2=**fopen**("../convertitore_alfabeto/fileUTF/".$nomefile2.".UCT","w");

Questa funzione apre un file restituendo il puntatore ad esso. Nel primo argomento è specificato il percorso del file, nel secondo la modalità di apertura, se in lettura, scrittura od aggiunta.

fputs($file2,$HTTP_GET_VARS[fono]);

Questa funzione inserisce il valore contenuto nel secondo argomento, nel file il cui puntatore è passato nel primo argomento.

fclose($file2);

Questa funzione chiude il file il cui puntatore è passato come argomento.

include("funzioni_synth/caricavettore.php3");

Questa funzione carica in memoria la funzione richiamata dal programma principale il cui percosrso è specificato nell'argomento.

```
<?if($dir=opendir("../convertitore_alfabeto/fileUTF")){
    while(false!==($file=readdir($dir))){
```

In questo caso si è utilizzato come esempio una parte di ciclo, per mostrare direttamente un utilizzo pratico: se la directory, il cui percorso è specificato nell'argomento della **opendir**, è aperta con successo, vengono caricati tutti i file presenti con **readdi**r finché questa funzione li trova al suo interno.

3.1.5 Istruzioni di sessione

session_start();

Questa espressione apre una sessione di lavoro.

session_register("vai","omo","vettoreomo","dueomo","vettoredistringhe","pas","login","user");

Questa espressione inserisce nel registro di sessione tutte le variabili che saranno utilizzate.

session_unregister("vettoreomo");

Questa espressione elimina dal registro di sessione la variabile specificata.

3.1.6 Interazione con mysql

$conn = **mysql_connect**() or **die** ("non c'e'");

Questa espressione apre una connessione per l'uso del db. In caso di esito positivo restituisce l'indicativo di connessione altrimenti visualizza un messaggio di errore tramite la funzione **die**.

mysql_select_db("GEMMA",$conn) or **die** ("no db");

Questa espressione seleziona il db con l'indicativo di connessione ottenuto in precedenza. In caso di esito negativo visualizza un messaggio di errore tramite la funzione **die**.

$sql="**SELECT * FROM prefissisuffissi WHERE tipo='s'** ";
$result = **mysql_query**($sql,$conn);

La prima espressione è un esempio di comando SQL mentre la seconda è la funzione che invia la query al db utilizzando l'indicativo di connessione. Ritorna una variabile booleana indicante l'esito dell'operazione.

while($suffisso=**mysql_fetch_row**($result)…

E' indicato un frammento di codice per indicare che se $result è diversa da zero, esito positivo della query inviata, viene eseguita l'operazione di fetch, cioè caricamento, dei dati. Una modalità alternativa è **mysql_fetch_array,** che carica direttamente in un array i dati prelevati.

3.1.7 Espressioni regolari

Un'espressione regolare può essere considerata a tutti gli effetti un micro linguaggio inserito all'interno del linguaggio stesso, permettendo di effettuare una serie di controlli, anche molto complessi, con una stringa di codice estremamente compatto e leggibile. Come accennato troveremo due nomi per descrivere le espressioni regolari in **PHP**: le **REGEXP**, contrazione di regular expression, proprie del **PHP**, le cui istruzioni hanno la parola chiave **ereg**, e le **PCRE**, cioè **Perl Compatible REGEXP**, le cui istruzioni hanno la parola chiave **preg**. Queste ultime sono state le più usate nel presente lavoro poiché presentano un grado di flessibilità maggiore rispetto alle altre. Un'espressione regolare permette di individuare un carattere od una sequenza di caratteri specifica all'interno di una stringa, restituendo una variabile di tipo booleano in risposta. Ad esempio se volessimo verificare la presenza della sequenza di caratteri "io" all'interno della parola gioco scriveremmo:

$parola="gioco";

ereg("io",$parola);
preg_match("/io/",$parola);

In questo semplice caso l'istruzione verifica se all'interno della variabile parola sia presente la sequenza di caratteri io: poiché alla variabile è stato assegnato il valore gioco l'espressione regolare darà un esito positivo, viceversa si avrebbe nel caso parola fosse stata giacca. Nel caso sia necessario memorizzare la sequenza, o pattern, riscontrata all'interno della variabile in esame, è possibile introdurre una variabile di appoggio, che chiameremo solo per comodità **$match**, nella quale viene memorizzato un vettore, contenente l'intera stringa nel primo elemento e nei successivi gli eventuali pattern trovati. Per attivare la funzione di memorizzazione è necessario però inserire fra parentesi tonde la parte di espressione che eventualmente interessa. Riprendendo l'esempio precedente, e non sapendo se a priori la parola che contenga io sia gioco o giocare, potremo scrivere la seguente espressione:

preg_match("/io(\w+)/",$parola,$match);

all'interno del primo elemento di $match troveremo **co** o **care** a seconda del valore di parola. Oltre queste semplici operazioni di matching tra variabili è anche possibile modificare un'intera sequenza di caratteri con altre od addirittura, vettore di caratteri, tramite l'espressione preg_replace. Di seguito riportiamo un esempio:

$parola=preg_replace("/io/","e",$parola);

questa istruzione sostituisce nella variabile parola la sequenza di caratteri io con la sequenza costituita dal solo carattere e: si noti che l'operazione non sarebbe visibile senza una preventiva assegnazione del risultato della stessa ad una variabile che, in questo caso, è la stessa sulla quale si opera il controllo. Nulla avrebbe vietato di riassegnare il risultato ad un'altra variabile lasciando quindi immutata quella originale. Tramite l'uso di operatori specializzati, è possibile controllare la presenza della sequenza di caratteri in testa alla variabile, in coda od in qualsiasi posizione; controllare ad esempio il numero di ripetizioni oppure la possibilità di combinarla con altri caratteri. Questi operatori sono utilizzabili con ciascuna delle espressioni regolari implementate nel linguaggio. Sono riportati nell'elenco seguenti le espressioni regolari utilizzate in questo lavoro, che rappresentano solo una parte di quelli implementati

dal linguaggio. Per ulteriori chiarimenti si rimanda al manuale.

preg_match("/(cg)([aeou])/i",$sequenza,$match);

In questa espressione si verifica se vi siano nella stringa $sequenza, caratteri *c, g* seguiti da una vocale: all'interno delvettore $match troveremo nell'elemento di indice uno il valore riscontrato nella prima parentesi ed analogamente per il secondo. Nel vettore di indice zero avremo tutta la sequenza riscontrata.

$stringa=**preg_replace**("/grafema/i","fonema",$stringa2);

Se nella variabile $stringa2 è riscontrata la sequenza grafema, la si sostistuisce con fonema ed il risultato è posto nella variabile $stringa.

$simboliappoggiotre=array("/ó/","/è/","/é/","/ò/");
$simbolifoneticitre=array("o","e","e","o");
$tre=**preg_replace**($simboliappoggiotre,$simbolifoneticitre,$tre);

Questa funzione è analoga alla precedente ma utilizza un vettore di variabili.

Questa panoramica sulle funzioni principalmente usate nel programma dovrebbe fornire gli strumenti necessari per la comprensione dei codici riportati nel manuale operativo del **CRIiSTAL**, dove sono esaminati tutte le funzioni ed i programmi del progetto.

3.2 GLI ALTRI SOFTWARES UTILIZZATI

3.2.1 APACHE

Come già accennato, il **PHP** è un linguaggio lato server, e come tale ha bisogno di un programma per il trasferimento dei dati tramite protocollo **http,** cioè un Web server e la possibilità, quindi, di inviare dati da una pagina all'altra durante la navigazione tramite il browser. Nello specifico è stato utilizzato il programma **Apache**. In rete è inoltre possibile reperire documentazione e faq per la sua installazione e configurazione.

3.2.2 MYSQL

La base di dati, è una struttura che permette di immagazzinare dati ordinati secondo un determinato criterio, come si farebbe ad esempio in un negozio, per realizzare l'inventario dei prodotti disponibili in un magazzino. All'interno di questa base di dati, è necessario creare delle tabelle, per poter catalogare dei dati aventi delle caratteristiche comuni: nell'inventario di un negozio di alimentari cercheremo di ordinare i prodotti in base alla loro deperibilità, o della provenienza, o del tipo. All'interno di ogni tabella poi, specificheremo, utilizzando come campo di ricerca il nome del prodotto, una serie di informazioni che potrebbero essere il prezzo, il peso ecc.

Una organizzazione logica e razionale del database permette di ottenere buone prestazioni generali in quanto la ricerca sarà effettuata nel minor tempo possibile. Per questa applicazione specifica, uno qualsiasi dei database disponibile, sarebbe andato bene. Fra i più utilizzati in coppia con il **PHP** troviamo il **Postgres** ed il **Mysql**. Fra i due è stato scelto quest'ultimo perché, pur essendo meno versatile a livello di operazioni su db, risulta estremamente veloce ed implementa comunque, tutte le funzionalità necessarie al progetto. Considerando che la base di dati potrebbe essere in continuo aggiornamento, e di conseguenza aumenterebbero i tempi di accesso all'intero sistema, si è preferito privilegiare quest'aspetto. Naturalmente inserire all'interno del codice dati presenti nel db avrebbe sicuramente velocizzato l'intero sistema ma non è pensabile poter inserire centinaia di vocaboli con le loro proprietà all'interno delle varie funzioni poiché qualsiasi aggiornamento della lista di vocaboli, avrebbe richiesto la modifica del codice. Con una base di dati esterna invece, è possibile aggiornare semplicemente quest'ultima indipendentemente dal programma, ed in più creare una procedura di correzione automatica, che permetta di inserire ogniqualvolta si riconosca un errore nella trascrizione della parola immessa, il dato corretto all'interno della lista.

Vengono di seguito riportate le istruzioni del linguaggio **SQL** utilizzate nel software rimandando invece all'organizzazione della base di dati al manuale operativo del **CRIiSTAL**. Si faccia attenzione all'uso dei due tipi di virgolette, necessari al corretto funzionamento della query.

SELECT * FROM `apertechiuse` WHERE grafema= '$parola'

Questa espressione permette di effettuare una selezione degli elementi nella tabella specificata dopo la parola **FROM**, dove l'elemento **grafema** assume il valore contenuto nella variabile **$parola**. L'asterisco dopo **SELECT** indica su tutti gli elementi.

DELETE FROM `apertechiuse` WHERE `grafema`= '$parola'

Questa espressione permette di effettuare una cancellazione nella tabella specificata dopo la parola **FROM**, dove l'elemento **grafema** assume il valore contenuto nella variabile **$parola**.

INSERT INTO `apertechiuse`(`grafema`,`tipo`,`posizione`)VALUES('$parola','$tipo','$posizione')

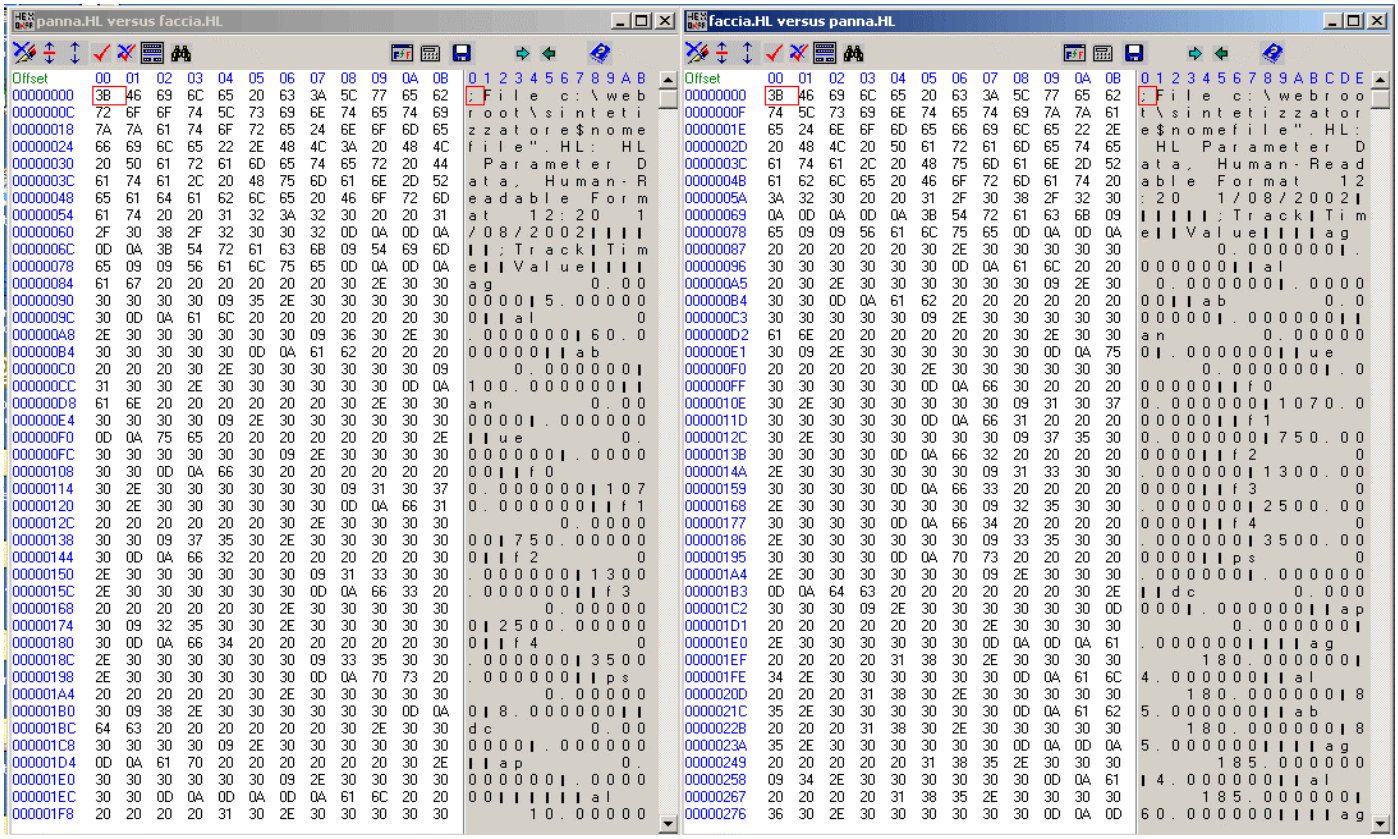
Questa espressione permette di effettuare l'inserimento degli elementi contenuti dopo la parola **VALUE**, nella tabella specificata dopo la parola **INTO**, nei campi indicati dopo il nome della tabella.

3.2.3 Spyguru

Questo piccolo programma, liberamente distribuito in rete, è un editor esadecimale che permette di visualizzare i codici relativi ad ogni simbolo all'interno di un file. Come visto in precedenza, il sintetizzatore **HLSyn** può comunicare con l'esterno solo attraverso dei files di testo opportunamente formattati: La procedura seguita per la decodifica è stata la seguente:

1) è stato generato un file di testo dal sintetizzatore
2) si è ricreato un file apparentemente uguale in base alla visualizzazione dal monitor
3) tramite la modalità versus di **Spyguru** sono stati aperti entrambi i files e si è effettuato il confronto simbolo per simbolo
4) ad ogni codice esadecimale è stato associato il relativo simbolo

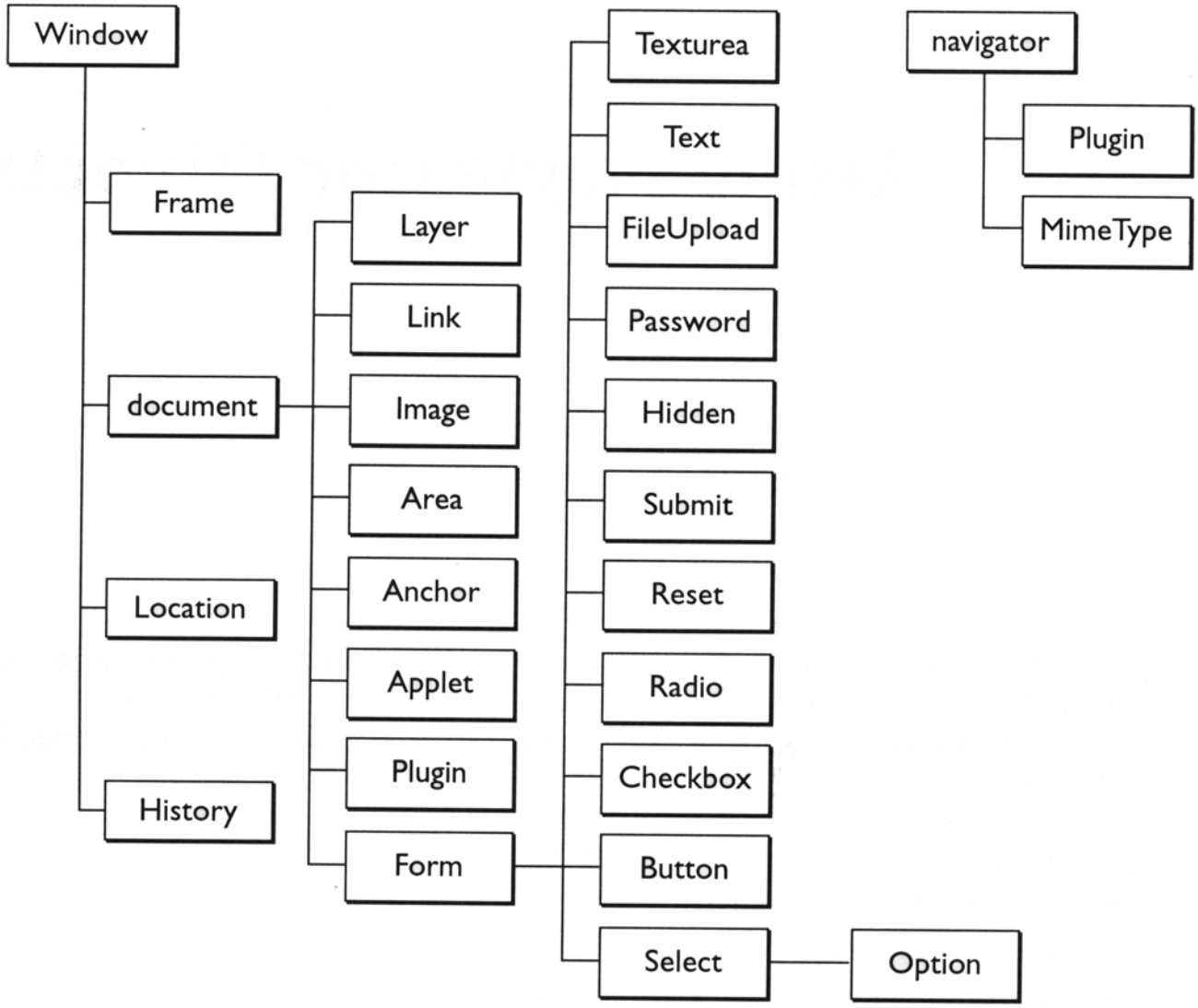
La tabella con le corrispondenze è riportata nel **CRIiSTAL REFERENCE**.
Di seguito mostriamo un esempio della modalità versus.



3.2.4 JavaScript

Il **JavaScript**, che come abbiamo visto è un applicativo lato client, è stato utilizzato per rendere interattive le pagine di immissione dati e poter inserire correttamente le informazioni necessarie alla corretta esecuzione del programma. E' possibile infatti attivare determinate funzioni di inserimento dati solo in conseguenza di altre azioni, come la pressione di un bottone o la selezione di una **checkbox**. Nel caso della procedura di correzione, infatti, è necessario indicare se la parola in esame appartenesse alla categoria omografi per poterla inserire opportunamente nella base di dati. Nel caso della sintesi, invece, occorre poter specificare il nome del file da creare solo dopo aver selezionato l'opzione corrispondente, per non generare files vuoti ad ogni invio dati.

Il **JavaScript** tratta tutti gli attributi **HTML** di una pagina come degli oggetti, associando loro delle proprietà e delle operazioni di modifica. Questa associazione avviene secondo il seguente schema ad albero:



Ogni proprietà è accessibile con la seguente sintassi:

document.correzione.elements["attivae_<? echo $stringa2[1];?>"].disabled=true;

Ad ogni punto corrisponde la selezione di un ramo dell'albero e l'ultima istruzione è la responsabile dell'assegnazione del valore desiderato. Nelle parentesi quadre è indicato il nome dell'elemento a cui ci si riferisce: in questo caso il nome è creato da codice **PHP**. E' possibile definire delle funzioni richiamabili da particolari eventi **HTML**, gli **event handlers** associati ad operazioni compiute dall'utente nell'interazione con il browser, come la selezione di una casella, la pressione su un buttone ed altro. Nel manuale di riferimento è possibile consultare l'elenco completo.
Un esempio di uso di uno di questi **event handlers** è il seguente:

<INPUT type="checkbox" name="attivai_<? echo $stringa2[1];?>" onclick="stacca<?echo $indexdis; ?>('1','1');" >

Nel momento in cui viene selezionata la casella relativa della **checkbox**, quindi con un "**click**" sull'oggetto in esame, l'evento **onclick** è attivato ed invoca la funzione "**stacca**" con gli opportuni parametri. Questa funzione è utilizzata nelle pagine di correzione e si rimanda al **CRIiSTAL MANUAL** per vederne il funzionamento e gli effetti. Un altro uso dell'**onclick** si trova nella pagina dell'ascolto degli esempi **Wave** mentre nella pagina di scelta files **UTF** è usato l'evento **OnChange**.
Per ulteriori approfondimenti si rimanda al manuale del linguaggio.

3.2.5 PHP-MYADMIN-2.2.0

Questo applicativo permette di utilizzare un'interfaccia di tipo **WEB** per interagire con il db senza dover utilizzare le righe di comando, permettendo una più veloce operatività ed una visione completa delle modifiche apportate.