UNIVERSITA' DEGLI STUDI DI PADOVA

FACOLTA' DI SCIENZE STATISTICHE

CORSO DI LAUREA SPECIALISTICA IN STATISTICA E INFORMATICA

TESI DI LAUREA

Estrazione ed elaborazione di contenuti informativi dal web

Relatore:
Prof. GRAZIANO DEAMBROSIS

Laureando: Tomaso Minelli
Matricola: 513997

Anno Accademico 2005/2006

Contenuti

Prefazione

"[...] l'eccesso di comunicazione, il pancomunicativismo esasperato, decreta l'eclissi stessa della comunicazione."[1]

Già da qualche anno, per esigenze più di trasparenza che di comunicazione vera e propria, molti enti mettono a disposizione "in tempo reale"[2] agli utenti, alla cittadinanza, le informazioni di cui sono in possesso: ne sono un esempio gli indici della Borsa Italiana, inseriti con un ritardo di quindici minuti nella loro pagina principale[3], la Polizia di Stato che riporta giorno per giorno le statistiche degli incidenti[4] o i siti che riportano l'"impact factor" di tutti gli articoli scientifici pubblicati.

Queste informazioni sono esposte sul Web in maniera fruibile dagli utenti, ma non sempre sono facilmente accessibili dataset o basi di dati con le informazioni stesse, perciò qualsiasi analisi che esuli dalle politiche e dalle esigenze degli enti che le espongono risultano impossibili; a volte questi dati sono esposti in maniera puntuale per

1 Informazione e Conoscenza: l'incertezza creativa. Pubblicato in "Atti dell'Istituto Veneto di Scienze, Lettere ed Arti", Classe di Scienze Fisiche, matematiche e naturali, Tomo 160, II-III, 2002. Antonella Artista http://www.scienzeformazione.unipa.it/docenti/artista/artistainfoconoscenza.pdf

2 Fornire le informazioni in "tempo reale" denota il fatto di renderle fruibili non appena queste vengono acquisite; il termine ha assunto poi una connotazione più generica, riferendosi a tutte quelle operazioni eseguite in tempi molto ravvicinati, non necessariamente istantanee o sincrone.

3 http://www.borsaitalia.it/homepage/homepage.htm

4 http://www.poliziadistato.it/pds/stradale/archivio/index.php

breve tempo, per poi essere presentati in maniera aggregata per periodi antecedenti, (ad esempio i dati della Borsa Italiana esposti in dettaglio solo per la giornata corrente); alcuni dati sono, inoltre, accessibili solo attraverso procedure di autenticazione che rendono l'analisi degli stessi difficilmente effettuabile con “crawler”[5] tradizionali, ad esempio i dati sull'impact factor. Che questo tipo di problematiche sia molto sentito viene dimostrato dal fatto che “crawler” ed “information retrieval” sono due dei più diffusi argomenti di pubblicazione scientifica in questo momento[6].

Nelle pagine seguenti si vuole sviluppare lo studio di una serie metodologie per acquisire, archiviare e rendere graficamente, dati estrapolati da pagine web: verranno presentati, innanzitutto, dei casi di studio reali, nel prossimo capitolo, per poi analizzare le tecnologie allo stato dell'arte per acquisire documenti dal web (“Acquisizione Dati”), per estrapolarne le informazioni (“Estrapolazione delle Informazioni”), per archiviarli in maniera opportuna (“Archiviazione dei dati con Round Robin Database”) e per rendere i dati in maniera grafica (“Grafici con RRDTool”); si discuterà poi, brevemente, di alcuni sistemi per automatizzare l'acquisizione (“Schedulazione delle attività”); infine verrà presentato un tool, chiamato WebAcquire, che mette insieme le tecnologie studiate[7] per acquisire, archiviare e rendere

5 Per crawler si intendono tipicamente strumenti che operano iterativamente, scaricando una pagina Web, processandola e seguendo i link per trovare altre pagine da scaricare e il risultato di questa operazione è una collezione di documenti da memorizzare sul database; il nostro strumento non contemplando l'iteratività si discosta da questa definizione.

6 Effettuando una ricerca su “google scholar” (http://scholar.google.com) troviamo, con data di pubblicazione successiva al 2000, inserendo come chiave di ricerca crawler 3,550 risultati, mentre inserendo “information retrieval” ne troviamo 230,000. Come confronto inserendo “ordbms” otteniamo 451 risultati.

7 Si predilige l'utilizzo di tecnologie “libere”, nell'accezione più ampia del termine, ovvero dagli stumenti con licenza GPL, ovvero open source o free-software, agli strumenti BSD, una licenza meno restrittiva che consente anche la vendita del codice, ed a qualsiasi realtà che permetta il libero utilizzo del codice sorgente, la loro modifica e personalizzazione.

graficamente dati disponibili nel web.

L'idea di questo studio, inoltre, è di presentare e valorizzare alcuni strumenti nati per compiti specifici ma implementati in maniera così flessibile da risultare particolarmente adatti ai nostri scopi; ad esempio WWW::Mechanize, insieme ad HTTP::Recorder, nascono come strumenti per il debugging di web-application mentre RRDTool nasce come software per l'analisi di rete e per il controllo di server applicativi[8]: l'obbiettivo che ci siamo posti è di dimostrare la validità delle due librerie per l'acquisizione di dati, e di dimostrare come l'acquisizione di serie storiche abbiamo nella tecnologia dei Round Robin Database un valido alleato, purché la tecnologia sia utilizzata in maniera adeguata.

Ogni capitolo, ad esclusione del primo, presenterà una panoramica della tecnologie specifiche, quindi presenterà la soluzione adottata per alcuni case study proposti nel primo capitolo; l'ultimo capitolo tratterà l'implementazione del software WebAcquire e come utilizzarlo, o meglio configurarlo, per risolvere in maniera omogenea gli esempi proposti.

Per non appesantire il testo, nelle prime due appendici sono scritti due approfondimenti su RRDToll e su Mechanize, rispettivamente: essendo i due software più corposi trattati non si voleva distrarre il lettore con trattazioni troppo specifiche.

Le ultime tre appendici contengono, infine, il codice PERL degli esempi dei primi capitoli, di alcuni script di utilità presentati ed in ul-

8 Sviluppato inizialmente come un tool PERL per l'archiviazione di dati esclusivamente per numeri interni come libreria del software di controllo di rete MRBS(?), è stato poi sviluppato come progetto isolato accettando anche numeri reali, disponendo di archivi in formato binario e di codice in linguaggio C.

timo lo script WebAcquire.

Come strumento di programmazione di riferimento si è scelto di utilizzare il linguaggio PERL: utilizzare un linguaggio di scripting incide chiaramente sulle “performance” del software ma, d'altra parte, semplifica e rende più didatticamente utile, l'analisi e la correzione del software, nonché la sua diffusione allorché si adottino tipi di licenza “liberi”, ovvero che lo si renda disponibile in rete; i motivi che inducono ad usare il PERL come linguaggio di sviluppo sono anche legati al tipo di compito che ci interessa: questo linguaggio è profondamente legato alla rete (è stato il primo con il quale si sono sviluppate le pagine web “dinamiche”) ed il suo acronimo[9] lo rende il candidato ideale per l'estrazione di contenuti dai documenti; infine è disponibile in PERL il set di librerie Mechanize: questo strumento nasce per effettuare debugging e analisi dei propri siti web, nonché per verificarne le prestazioni; essendo uno strumento nato per il debugging non eccelle nei tempi di risposta ma abbiamo già accantonato in precedenza il problema delle performance del nostro software.

9 PERL: Practical Extraction and Reporting Language ovvero linguaggio pratico di estrazione e presentazione. I maligni sostengono che l'acronimo sia Pathologically Electronic Rubbish List, ma il suo autore, Larry Wall smentisce l'affermazione con fermezza e qualche sorriso.

1. Case Study

Per chiarire l'uso e lo scopo degli strumenti che andremo a proporre, ci mettiamo di fronte a dei casi reali che ci permettano di mettere a fuoco le problematiche e le finalità del nostro lavoro: gli esempi esposti in seguito cercheranno di mettere in evidenza problematiche diverse legate all'acquisizione ed alla fruibilità dei dati on-line.

Sostanzialmente ogni esempio metterà l'utente di fronte a due problematiche distinte:

1. dati disponibili “one-shot”, ovvero disponibili per un limitato periodo di tempo e poi non più fruibili
2. dati disponibili ma assenza di schemi grafici riassuntivi che li rendano facilmente leggibili dagli utenti

Almeno una di queste due problematiche è presente in ognuno degli esempi presentati ma, più spesso, si trova un misto di queste problematiche, ad esempio si trovano degli schemi grafici riassuntivi troppo sintetici o i dati sono disponibili per tempi medio-lunghi, ovvero dati disponibili istantaneamente e poi reperibili solo in forma aggregata.

1.1 Dati della Polizia Stradale Italiana: incidenti, contravvenzioni e pattuglie impiegate

La sezione della Polizia Stradale (in seguito polstrada) della Polizia di Stato Italiana (in seguito polizia), mette a disposizione del cittadino i dati dettagliati, giorno per giorno, degli incidenti nella strade e delle contravvenzioni elevate[10], distinguendo i dati fra strade e autostrade, fra le varie tipologie di incidenti (mortali, con feriti o solamente con danni) e di contravvenzioni.

I dati esposti sono, come si vede dall'immagine, molto dettagliati e chiari, sono disponibili giorno per giorno a partire dal primo marzo del 2001; i dati possono essere anche acquisiti in maniera aggregata per intervalli di tempo attraverso un form in cui si possono specifica-

10 http://www.poliziadistato.it/pds/stradale/archivio

re il periodo ed i dati da aggregare[11].

Il cittadino ha a disposizione un quadro numerico molto approfondito e con un veste grafica chiara e facilmente leggibile che, purtroppo, non è affiancata da nessun tipo di diagramma grafico o indice statistico che rappresenti incrementi, decrementi o stagionalità dei fenomeni.

Il compito che ci prefiggeremo in questo caso è di acquisire tutti i dati resi disponibili dal primo marzo 2001 e quindi di acquisire i dati successivi giorno per giorno, esponendoli attraverso grafici che consentano di visualizzare in tempo reale l'andamento delle singole tipologie di dati acquisiti; i dati dovranno essere anche archiviati in forma tabellare per eventuali indagini statistiche più approfondite, atte, ad esempio, ad analizzare il vero effetto della patente a punti o il rapporto fra incidenti e presenza sul territorio di squadre della polstrada.

11 I dati sul numero di pattuglie impiegate è disponibile solo a partire dal primo settembre 2004

1.2 Dati della Borsa Italiana: valori degli indici di riferimento

Il sito della Borsa Italiana[12], a differenza della polstrada, mette a disposizione oltre ai dati numerici dei valori giornalieri dei suoi indici di riferimento, anche semplici grafici collegati con l'andamento tendenziale degli stessi.

12 http://www.borsaitalia.it

Oltre che degli indici principali, i dati sono esposti anche per ogni singola società quotata, fornendo quindi un buon esempio di come si possano rendere disponibili ai cittadini, ed in questo caso anche agli investitori, dati dettagliati, precisi e chiari.

Alla voce “interday” si possono, infine, visualizzare i valori minuto per minuto degli indici nella giornata corrente: questi valori potrebbero essere utilizzati per calcolare una eventuale stagionalità presente durante la giornata di apertura della borsa, per vedere se ci sono momenti migliori nella giornata per effettuare compravendite.

In realtà non è possibile individuare eventuali inter-stagionalità poiché non sono disponibili grafici dettagliati che registrino con intervalli temporali (ad esempio di cinque minuti) l'andamento giornaliero tendenziale. Per tale ragione ci prefiggiamo il compito di recuperare i dati di alcuni indici della borsa ogni minuto per crearne un archivio dettagliato (della “serie temporale”) su cui fare delle analisi approfondite e per creare dei grafici puntuali che non rappresentino solamente l'andamento tendenziale, ma che rappresentino dettagliatamente i movimenti nella giornata in corso e nelle giornate precedenti.

1.3 ARPA Emilia Romagna: dati sull'inquinamento atmosferico

Rispetto ai due casi precedenti, i dati esposti dalla Agenzia Regionale Prevenzione e Ambiente dell'Emilia Romagna nel loro sito sono fruibili solo in maniera molto limitata: non solo non sono disponibili grafici sull'andamento tendenziale o dettagliato dei dati, ma i dati numerici non sono accessibili neppure per più di alcuni giorni e sono inseriti con qualche giorno di ritardo nel sito[13].

Dati ambientali a cura di ARPA Emilia-Romagna

arpamo arpaweb

PM10 | O_3 | NO_2 | C_6H_6 | CO | SO_2

Rilevamento dell'inquinamento atmosferico nella provincia di **Modena**

Dati del giorno: 13/04/2006

In colore arancio sono evidenziati i valori oltre il limite di legge — Come si legge il bollettino

Particolato < 10µm (PM10)		Media 24 ore (µg/m³)	Superamenti VL (50 µg/m³)	Superamenti consentiti	Conforme DM60/02
Agglomerato R4 - MO	CARPI - CARPI 2 (VIA REMESINA)	35	39	35	No
	MODENA - MO - PARCO FERRARI (PARCO FERRARI)	23	0		
	MODENA - MO - VIA GIARDINI (VIA GIARDINI)	38	54		
	MODENA - MO - VIA NONANTOLANA (VIA CIMONE)	42	49		
Agglomerato R5 - MO	FIORANO MODENESE - SPEZZANO 2 (VIA MOLINO)	28	49	35	No

torna all'inizio

Ozono (O_3)		Media oraria max (µg/m³)	Ora valore Max	Superamenti (180 µg/m³)		
Agglomerato R4 - MO	MODENA - MO - VIA NONANTOLANA (VIA CIMONE)	51	15	0		
Agglomerato R5 - MO	FIORANO MODENESE - SPEZZANO 1 (VIA CANALETTO 80)	79	16	0		
	MARANELLO - MARANELLO (VIA SPERI)	106	16	0		
Zona A - MO	MIRANDOLA - MIRANDOLA (VIA DANTE ALIGHIERI)	49	16	0		
		Media 8 ore max (µg/m³)	Ora valore Max	Superamenti VL (120 µg/m³)	Superamenti consentiti	Conforme DL183/04
Agglomerato R4 - MO	MODENA - MO - VIA NONANTOLANA (VIA CIMONE)	43	19	0	25	Sì
Agglomerato R5 - MO	FIORANO MODENESE - SPEZZANO 1 (VIA CANALETTO 80)	69	18	0	25	Sì
	MARANELLO - MARANELLO (VIA SPERI)	85	19	0		
Zona A - MO	MIRANDOLA - MIRANDOLA (VIA DANTE ALIGHIERI)	43	18	0	25	Sì

torna all'inizio

Biossido di Azoto (NO_2)		Media oraria max (µg/m³)	Ora valore Max	Superamenti VL + MT (240 µg/m³)	Superamenti consentiti	Conforme DM60/02
	CAMPOGALLIANO - CAMPOGALLIANO (VIA KENNEDY)	131	21	0		
	CARPI - CARPI 1 (VIA CARLO MARX)	128	23	0		
	CARPI - CARPI 2 (VIA REMESINA)	178	20	0		

13 http://service.arpa.emr.it/qualita-aria-2005/bollettino.aspx?prov=mo

In questo caso il nostro compito sarà, come nei casi precedenti, di creare un archivio dei dati fruibile per indagini più approfondite, attraverso la generazione di grafici dettagliati e tendenziali degli andamenti.

A parziale giustificazione dell'ARPA dell'Emilia Romagna[14], è bene sottolineare che i dati Veneti dettagliati sono, ad oggi, accessibili solamente per valori aggregati; inoltre, mentre scriviamo queste righe, non siamo riusciti a recuperare i dati poiché il collegamento "saltava" regolarmente restando disattivo anche per alcuni giorni consecutivi. Per le regioni Piemonte e Lombardia, pur non avendo riscontrato problemi di collegamento, i dati sono presentati con molte lacune temporali e locali e sempre in indici aggregati su cui non è possibile fare nessuna considerazione ed analisi ulteriore: perciò alla fine abbiamo scelto la regione Emilia Romagna perché risultava comunque la più dettagliata e, inoltre, presentava un interessante controllo di accesso, molto utile per i fini della nostra trattazione.

14 E' decisamente inquietante, inoltre, notare che collegandosi alla pagina: http://service.arpa.emr.it/qualita-aria-2005/ si presenta una pagina intitolata "Correzione manuale dati di qualità dell'aria"

1.4 In sintesi

Il litemotive che unisce gli esempi esposti è la creazione di grafici ed analisi che non sono disponibili direttamente nel sito web che ci fornisce i dati; potremmo dire che l'assenza dell'informazione nel sito di riferimento può mancare per necessità di sintesi (come nel caso del sito della Borsa Italiana), per diversità di intenti o per mancanza di know-how (come nel caso della polstrada che non espone alcun tipo di grafico), o addirittura per motivi non troppo chiari (come nel caso dell'ARPA dell'Emilia Romagna); ricordiamo che dopo l'analisi dei siti delle ARPA delle regioni italiane analizzate, abbiamo preso in considerazione l'Emilia Romagna in quanto è l'unico sito che espone in maniera chiara i propri dati.

L'unica vera perplessità suscitata dal sito emiliano è che collegandosi alla pagina `http://service.arpa.emr.it/qualita-aria-2005/`, come si evidenziava nella nota del precedente paragrafo, compare una finestra di autenticazione che permette di entrare nell'area “Correzione manuale dati di qualità dell'aria”[15], fatto decisamente inquietante.

15 Per chiedere la password si è invitati a scrivere a: cghirelli@sc.arpa.emr.it, fscrepanti@sc.arpa.emr.it

Correzione manuale dati di qualità dell'aria

2. Acquisizione dati

In questo capitolo ci occuperemo di alcuni software e librerie per l'acquisizione dei dati dalle pagine web, nati per soddisfare diverse necessità:

3. **wget** è uno software a linea di comando nato per semplificare il download di grosse moli di dati attraverso i protocolli http: supporta il “resume”[16] dei download ed anche la possibilità di effettuare “mirroring”[17] di interi siti web;
4. **CURL** è sia un tool a linea di comando, che una libreria per il download di singoli documenti e permette di personalizzare la richiesta inserendo anche variabili di sessione, cookie e simulando l'invio di form html;
5. **WWW::Mechanize** è una libreria PERL nata per effettuare debugging completi dei propri siti web; oltre a tutte le caratteristiche di CURL permette di effettuare richieste successive mantenendo attive le sessioni[18], simulando così in maniera completa la navigazione attraverso un browser classico.

16 Ovvero la capacità di riprendere lo scaricamento di un file di grosse dimensioni in un secondo momento, dopo una sua interruzione.
17 Ovvero la possibilità di scaricare ricorsivamente tutti i documenti (testo, immagini e quant'altro) che compongono un sito web, per farne una riproduzione fedele nel proprio disco locale che potrà essere consultata anche senza accedere nuovamente alla rete.
18 In alcuni casi particolari si può riuscire perfino ad eseguire codice javascript all'interno delle pagine in maniera vistuale.

2.1 Wget

Wget[19] è un progetto della GNU Foundation per acquisire file attraverso i più diffusi protocolli sul web: http, https ed ftp; è un prodotto molto semplice, snello e funzionale che permette, attraverso l'opzione “-c”, di continuare un download precedentemente completato ed attraverso l'opzione “-m” di effettuare il “mirror” completo di un intero sito web; oltre a queste due opzioni, wget ne espone molte altre di discreto interesse che permettono di definire l'user-agent (-U), di inviare alcuni header (--header) ed anche di inviare dati o file con il metodo post (--post-data, --post-file), benché non siano di facile utilizzo: visto che sia la polstrada che la Borsa Italiana usano solo chiamate get, potremmo utilizzare wget per acquisirne i dati.

```
wget -O polstrada.AAAAMMGG.html \
      -U “Mozilla/5.0 (X11; U; Linux i686; it-IT; rv:1.7.12) \
      Gecko/20051010 Firefox/1.0.7” \
      http://www.poliziadistato.it/pds/stradale/\
      archivio/zoom.php?vg=GG.MM.AAAA
```

Questo codice acquisisce la pagina con gli incidenti stradali di un giorno prefissato nel file polstrada.AAAAMMGG.html, simulando la navigazione con un browser “Mozilla Firefox” in ambiente Linux/X11; al posto di GG.MM.AAAA va inserita la data con otto caratteri.

Introduciamo qui l'utilizzo del carattere backslash (\) per indicare che il testo a capo è in verità parte della stessa linea e viene messo a capo solo per motivi di chiarezza espositiva.

19 http://www.gnu.org/software/wget

Alla stessa maniera potremmo acquisire i dati delle ore 16.30 della Borsa Italiana con il seguente comando:

```
wget -O borsa.`date +%Y%m%d`.16.30.html \
      http://www.borsaitalia.it/bitApp/\
      intraday.bit?isin=II999&target=indici&lang=it&\
      from=1630&querymode=byTime&HH=16&MM=30
```

Vediamo qui una delle problematiche che si riscontra utilizzando strumenti a linea di comando: per impostare il nome del file abbiamo dovuto eseguire del codice in linea attraverso l'operatore backtik (`): questo sistema, seppur valido, rende un nostro eventuale insieme di script utilizzabile solamente nei sistemi che espongano il comando date e che implementino l'uso dell'operatore backtick: questo è il motivo principale che ci fa scartare strumenti a linea di comando dalla nostra analisi e che ci farà, d'ora in poi, analizzare eclusivamente librerie multi-piattaforma, di preferenza disponibili per linguaggio PERL.

2.2 CURL

CURL[20] è disponibile sia a linea di comando che come libreria (libcurl) e, in questo formato, è utilizzabile anche in script PERL attraverso il wrapper WWW::Curl::easy.

Espone tutte le caratteristiche generali già viste in wget, ad esclusione del mirroring e della possibilità di continuare un download precedentemente iniziato, aggiungendo, però, una maggiore quantità di opzioni per inviare richieste con il metodo post, gestire i cookie e, insomma, poter creare delle richieste HTTP davvero complete.

La liberia WWW::Curl::Easy permette di effettuare richieste anche molto complesse, sfruttando il metodo `setopt` che permette di impostare un notevole numero di parametri nella connessione: vediamo come utilizzare questa libreria per recuperare i dati dei nostri esempi, partendo dalla polstrada:

```
use WWW::Curl::Easy;
my $GG='01';
my $MM='04';
my $AAAA='2006';
my $page_url="http://www.poliziadistato.it/pds/stradale/\
      archivio/zoom.php?vg=${GG}.${MM}.${AAAA}";

my $curl = new WWW::Curl::Easy;
$curl->setopt(CURLOPT_URL, $page_url);
$curl->perform;
print $curl->getinfo(CURLINFO_HTTP_CODE);
```

Mandato in esecuzione, lo script è assolutamente equivalente alla chiamata wget precedente (tranne per il fatto che invia il risultato direttamente in output e non lo salva su file): è sufficiente modificare

20 http://curl.haxx.se/

la variabile `$page_url` per adattare l'esempio anche alla Borsa Italiana. Per ottenere i dati dell'ARPA Emiliana dobbiamo invece integrare nel codice alcune opzioni che simulino l'invio del form di ricerca presente nella pagina: per fare ciò utilizzeremo l'opzione curl CURLOPT_POSTFIELDS:

```
use WWW::Curl::Easy;
my $GG='18';
my $MM='04';
my $AAAA='2006';
my $page_url="http://service.arpa.emr.it/\
      qualita-aria-2005/bollettino.aspx?prov=mo";

my $curl = new WWW::Curl::Easy;
$curl->setopt(CURLOPT_URL, $page_url);
$curl->setopt(CURLOPT_POSTFIELDS, "__EVENTTARGET=drpData");
$curl->setopt(CURLOPT_POSTFIELDS, "__EVENTARGUMENT=");
$curl->setopt(CURLOPT_POSTFIELDS, "__VIEWSTATE=");
$curl->setopt(CURLOPT_POSTFIELDS, "drpData=${GG}/${MM}/${AAAA}");
$curl->perform;
print $curl->getinfo(CURLINFO_HTTP_CODE);
```

Questo script in verità visualizza sempre la pagina web contenete i valori dell'ultimo giorno disponibile ed ignora i parametri inseriti come postfields: vedremo nella prossima sezione come eludere questo problema.

Le differenze fra wget e curl sono, come si può desumere, molto sottili[21] e risiedono principalmente nella ricchezza di opzioni disponibili in curl: in verità per i nostri scopi la ricchezza di opzioni disponibili non è il vero motivo della scelta, bensì preferiamo quest'ultimo per la possibilità di utilizzarlo come libreria e quindi di creare script più portabili fra diversi sistemi; curl però in alcuni casi non è sufficiente, come si deduce dall'ultimo esempio, perciò utilizzeremo anche la tecnologia WWW::Mechanize esposta nel prossimo paragrafo.

21 Una buona descrizione delle differenze fra i due prodotti si può leggere da: http://www.samhart.com/cgi-bin/classnotes/wiki.pl?UNIX01/Wget_And_Curl

2.3 WWW::Mechanize

Eseguendo l'ultimo script della sezione precedente ci rendiamo conto che il risultato non è la pagina web del giorno scelto, bensì la pagina web del giorno corrente: la scelta dei giorni si vuole che sia, infatti, strettamente limitata ai cinque giorni precedenti all'ultimo disponibile. Per fare ciò il programmatore ha predisposto un parametro (__VIEWSTATE), che viene valorizzato al momento del caricamento della pagina senza nessun parametro POST, e che viene controllato quando si cambia il giorno dalla lista presente al centro della pagina; per aggirare questo problema siamo costretti a simulare una “navigazione” utilizzando WWW::Mechanize: questo strumento, che nasce per effettuare debugging approfondito di applicazioni web, ci permette anche di effettuare la simulazione di navigazione succitata, per recuperare dati da siti web particolarmente ostici e che contengano diverse limitazioni per eludere i crawler e gli spider tradizionali.

Oltre al problema del campo VIEWSTATE, il form nella pagina viene inviato al server attraverso una funzione javascript attivata dalla scelta nella casella combinata: anche in questi casi WWW::Mechanize si rivela un utile compagno poiché permette anche di simulare script javascript nella pagina corrente.

Per concludere la presentazione di questo strumento la principale distinzione fra curl e Menchanize consiste nel fatto che il primo è pensato per chiamate “one-shot” mentre il secondo simula in pieno una navigazione, mantenendo attiva la sessione fra server e client durante la navigazione e, prima di inviare i form, modificando il valore solo dei campi di interesse, nonché eseguendo il codice javascript

legato a pulsanti o eventi qualora fosse necessario.

Risulta evidente che l'infrastruttura di Mechanize è quantomeno ridondante per i siti della polstrada e della Borsa Italiana, risulta invece decisiva per il sito dell'ARPA emiliana: vediamo qui di seguito come recuperare il giorno effettivamente richiesto:

```
use WWW::Mechanize;
$agent="Mozilla/5.0 (X11; U; Linux i686; it-IT; rv:1.7.12) ".
      "Gecko/20051010 Firefox/1.0.7";
$url="http://service.arpa.emr.it/qualita-aria-2005\".
      "/bollettino.aspx?prov=mo";

my $mech = WWW::Mechanize->new(
      agent=>$agent
);

$mech->get($url);
$mech->form_number(0);
$mech->select('drpData', '22/04/2006');
$mech->submit();

print $mech->content();
```

Notiamo come in questo caso non ci limitiamo a richiamare un url con la funzione get, bensì manipoliamo la pagina ricevuta e, dopo aver selezionato il primo form presente nella pagina, andiamo ad impostare il valore della combobox[22] drpData e quindi inviamo il form, esattamente quello che avremmo fatto attraverso un browser. Mechanize, inoltre, ci informa se stiamo eseguendo una operazione impropria: selezionando un giorno non disponibile, ad esempio, viene visualizzato il messaggio:

```
Illegal value '20/04/2006' for field 'drpData' at \
      /opt/local/lib/perl5/site_perl/5.8.8/WWW/Mechanize.pm line 1052
```

Questo ulteriore aiuto ci permette di limitare un altro problema

22 Le combobox, dette anche caselle a discesa, sono quegli strumenti di input grafico che visualizzano un elenco di voci da cui l'utente è obbligato ad effettuare una scelta.

che si presentava, ovvero che l'ARPA emiliana non visualizza i dati del giorno corrente, ma solo sei giorni a partire da alcuni giorni prima (a volte 2, ma anche 4 o 5): perciò sarà sufficiente eseguire un ciclo iterativo che scala dal giorno odierno all'indietro finché la funzione select non restituisce un valore vero[23]:

```
for (my $i=0;$i<100;$i++) {
      $date=ParseDate("oggi -$i giorni");
      if ($mech->select('drpData', UnixDate('%d/%m/%Y', $date)) {
            $mech->submit();
            print $mech->content();
            exit();
      }
}
print "Nessun giorno disponibile";
```

23 In PERL risulta come valore falso sia lo zero che il valore undef (restituito dalla funzione in caso di errore), mentre è vero tutto ciò che è diverso da zero, sia esso un numero, una variabile complessa o altro. Oltre a restituire undef la funzione chiama il metodo warn per segnalare l'errore all'utente.

3. Estrapolazione delle Informazioni: le espressioni regolari

Nel capitolo precedente abbiamo visto come acquisire le pagine web che ci interessano dalla rete, ora dobbiamo estrarre da queste pagine i dati che vogliamo archiviare ed utilizzare.

Un pagina HTML non è altro che un testo scritto in un linguaggio definito “a tag” che contiene tutta una serie di istruzioni per una visualizzazione gradevole dei contenuti e per il collegamento ad altri documenti immagini o altro: il nostro compito è di isolare le zone di nostro interesse e di estrarne i contenuti informativi; il problema principale che si pone nell'estrazione delle informazioni dalle pagine web è spesso dato dal fatto che le pagine non sempre mantengono la stessa struttura, bensì, come nel caso della Borsa Italiana, contengono zone variabili, destinate alla pubblicità che, se normalmente includono semplici immagini, a volte possono contenere anche pezzi di codice molto più complesso, anche con blocchi di codice javascript, che permettono l'apertura di finestre o la rotazione di vari messaggi pubblicitari[24].

Nel seguito vedremo alcune librerie PERL che permettono una analisi approfondita della struttura della pagina ma che risultano inadeguate e troppo complesse per il nostro scopo, per il quale si rivelerà essenziale l'utilizzo delle espressioni regolari, comunemente definite regexp dalla crasi della definizione inglese “regular expression”.

24 Si vedano i siti di repubblica (http://www.repubblica.it) e delle mappe Michelin (http://www.viamichelin.it) per avere una idea della varietà di inserti pubblicitari che possono comparire nelle pagine web: si valuti che con opportuni software di soppressione dei banner la home page del nostro quotidiano nazionale viene caricata anche dieci volte più velocemente.

3.1 Librerie per l'analisi sella struttura di una pagina web

In PERL sono state sviluppate numerose librerie per l'analisi delle pagine web per gli scopi più vari: dalla conversione della pagina HTML ad altri formati (HTML::Latex o HTML::WikiConverter) alla validazione del codice (HTML::Tidy), dalla gestione di template (HTML::Template) alla creazione di grafici (HTML::BarGraph). Per i nostri scopi possono tornarci utili quei tool che ci permettono una analisi sintattica e l'estrapolazione dei contenuti da una pagina web in forme più facilmente gestibili all'interno dei nostri programmi; non si pretende di dare in queste pagine una visione approfondita di questi strumenti, ma semplicemente di illustrare le potenzialità di analisi fornite da alcune di queste librerie.

3.1.1 HTML::Parser

Questa libreria permette di analizzare una pagina HTML in maniera molto simile ai più comuni parser XML: la classe viene inizializzata esponendo varie funzioni che saranno chiamate non appena ci troviamo di fronte ad un determinato tag o a del testo libero. Vediamo un piccolo esempio che ci illustra come estrarre il contenuto di tutti i tag “title” presenti in una pagina (tipicamente uno solo!):

```
use HTML::Parser ();

#funzione handler richiamata all'apertura di ogni tag
sub start_handler {
      my $tagname=shift;
      my $parser = shift;
      return if $tagname ne "title";
      $parser->handler(text => sub { print shift }, "dtext");
      $parser->handler(
            end  => sub { shift->eof if shift eq "title"; },
                  "tagname,self");
}

my $parser = HTML::Parser->new(api_version => 3);
$parser->handler(
      start => \&start_handler, "tagname,self");
#esegue il parsing del file passato come primo argomento
$parser->parse_file($ARGV[0]);
print "\n";
```

Come si nota dall'esempio, un'altra peculiarità di questa libreria è la possibilità di attivare degli hook[25] anche mentre il parsing è già attivo: questo significa che si può modificare il parsing della pagina allorché si incontra una sequenza di testo nota. I parametri passati alle funzioni attivate da HTML::Parser sono definiti nel secondo argomento del metodo handler: possono essere inviati anche gli attributi del tag (“attr” e “attrseq”) oppure il testo contenuto (“text”).

Dalla flessibilità di questa libreria ne consegue un difficile utilizzo in contesti specifici ma, d'altro canto, la rende uno strumento indispensabile utilizzato come base di partenza da molti altri parser, tra cui HTML::Entities, HTML::PullParser, HTML::TokeParser, HTML::HeadParser, HTML::LinkExtor, HTML::Form e HTML::TreeBuilder.

25 Con questo termine si definiscono quei costrutti che permettono di far si che una funzione sia richiamata allorché si presentano delle determinate condizioni, ovvero che il programma principale scateni quello che viene definito un evento: nella programmazione di interfacce grafiche si pensi che ogni azione dell'utente scatena un determinato evento, gestito dal programmatore associando un opportuno handler (maniglia) ad un hook del sistema.

3.1.2 HTML::TreeBuilder e HTML::Element

La classe TreeBuilder è derivata dalla classe HTML::Tree, a sua volta derivata dalla classe HTML::Parser esposta nel paragrafo precedente; una volta avviato il parsing questa classe può riempire delle strutture appartenenti alla classe HTML::Element, attraverso le quali si possono estrarre secondo una struttura ad albero i singoli tag che compongono la pagina, accedendo al loro contenuto ed ai loro attributi attraverso degli specifici metodi.

```
Use HTML::TreeBuider

my $tree = HTML::TreeBuider->new();

#Effettua il parsing del file passato come argomento
$tree->parse_file($ARGV[0]);

#visualizza il risultato del parsing
$tree->dump();

#trasforma la variabile in una classe HTML::Element contenete
#l'albero risultante
$tree->elementify();

#vislualizza il contenuto dell'albero creato
Dumper($tree);

#libera la memoria utilizzata
$tree->delete();
```

La struttura risultante dopo l'applicazione del metodo “elementify” è un albero piuttosto complesso che può essere sondato ed ispezionato attraverso alcuni metodi; content_array_ref permette di inserire la struttura ad albero in un array (i cui elementi possono essere a loro volta array) che può essere facilmente analizzato ed attraverso il quale si possono estrarre dati specifici. Per i nostri scopi risultano molto utili i metodi definiti come “secondary structural” ovvero di analisi più approfondita: tra questi risultano molto interessanti

find_by_tag_name che permette di estrarre i tag a seconda del loro tipo oppure find_by_attribute che ricerca gli elementi secondo i loro attributi.

Molto intreressante è anche il metodo tagname_map: il risultato sarà una variabile con struttura hash come la segunete:

```
{
      #Attraversa l'HTML con tutti i suoi sottoelementi
      'a'   => [ ...lista di tutti gli elementi 'a'... ],
      'em'  => [ ...lista di tutti gli elementi 'em'... ],
      'img' => [ ...lista di tutti gli elementi 'img'... ],
}
```

facilmente utilizzabile, ad esempio, per estrarre tutte le immagini, tutti i collegamenti o tutti gli indirizzi e-mail presenti in una pagina web; in alternativa si può utilizzare il più semplice extract_links per estrarre tutti i collegamenti della pagina relativi ad uno specifico tag, sia esso img, a o altro.

Risulta molto interessante, infine, il metodo address che ci permette di estrarre un singolo elemento dall'albero risultante, individuandone la posizione attraverso la più comune dot-notation:

```
$element=$tree->address(“0.10.3”);
```

estrae il terzo elemento del decimo ramo della struttura principale.

Malgrado la semplificazione che questa classe introduce è comunque molto raro riuscire ad analizzare pagine di cui non conosciamo la provenienza, che possono contenere strutture nidificate a contenuto variabile.

3.1.3 HTML::TableExtract

Normalmente i dati di nostro interesse in un documento HTML sono contenuti nelle celle di una tabella: TableExtract ci permette di estrarre da una pagina web esclusivamente il contenuto delle tabelle in essa presenti, trascurando tutto il contorno.

```
use HTML::TableExtract;

#Estrae la tabella che ha come intestazione Nome, Cognome e CF
$ts =  HTML::TableExtract->new(
      headers => [qw(Nome Cognome CF)]
);

foreach $t ($ts->tables) {
      foreach $row ($t->rows) {
            ($nome, $cognome, $cf) = @$row;
            print "Nome: $nome, Cognome: $cognome, ".
                  "Cofice Fiscale: $cf\n"
      }
}
```

La tabella di interesse può venire individuata, oltre che, come nell'esempio, per la sua intestazione, anche per la sua posizione in rapporto alle altre tabelle (dept), per il suo indice di posizione nella pagina oppure specificandone i suoi attributi.

Come si intuisce da questo esempio questa classe è molto più semplice da utilizzare rispetto ai metodi precedenti e permette di individuare molto facilmente delle aree di nostro interesse; restano comunque i rischi legati all'analisi, parsing, del codice HTML: questo linguaggio, derivato dal SGML, prevede la possibilità di minimizzare il codice (utilizzando ad esempio
 al posto di
) e di lasciare implicita la chiusura dei tag in alcuni casi (ad esempio, al ter-

mine di una cella, il tag </TD> chiude tutti i tag aperti dopo il tag di apertura <TD>); una conferma che il parsing del codice HTML non sia così semplice si ha dal fatto che quasi tutti i browser presenti sul mercato tendono a visualizzare il contenuto della medesima pagina in maniera leggermente diversa uno dall'altro, anche in casi in cui la pagina risulti conforme ai più stretti dettami imposti dal W3C[26].

26 Le verifiche che si possono effettuare sul codice HTML sono principalmente tre: la sua validità sintattica e semantica è verificabile con l'apposito servizio messo a disposizione all'url http://validator.w3.org; la correttezza dei fogli di stile che caratterizzano la pagina, è verificabile con il servizio on-line http://jigsaw.w3.org/css-validator/; infine si può verificare l'accessibilità della pagina, ovvero la sua conformità e facilità di utilizzo anche da parte di persone portatrici di handicap, principalmente di tipo visivo, che utilizzeranno degli strumenti ad-hoc per la navigazione. In questo ultimo caso non esiste uno strumento per la validazione del codice ma si può far ricorso alle linee guida presentate in http://www.w3.org/WAI/WCAG1AA-Conformance che distinguono anche il livello di accessibilità con la classe a singola o doppia A.

3.2 Le espressioni regolari

A differenza degli strumenti precedenti, le espressioni regolari permettono di sintetizzare in una singola riga di codice, ovvero in una semplice stringa di testo, ciò che, con altri strumenti di analisi testuale, potrebbe richiedere svariate righe di programma, aumentando così il possibile insorgere di errori.

Le espressioni regolari permettono di manipolare, inoltre, non solamente stringhe, bensì anche numeri e perfino dati binari attraverso opportuni adattamenti; poiché, come si è visto, il PERL nasce come strumento per i sistemisti in ambiente *nix, nei quali le informazioni e le configurazioni risiedono principalmente in file di testo, questo strumento doveva necessariamente permettere rapide analisi e modifiche delle impostazioni del sistema e rapide ricerche all'interno dello stesso.

Una espressione regolare serve ad individuare sia se all'interno di un testo compare una precisa sequenza di caratteri, sia se questo è stato scritto con una sintassi particolare oppure ad estrarre porzioni ben specificate; questo strumento permette anche di sostituire sezioni ben definite di un testo, nonché di eseguire una translitterazione[27] su di esso: questi ultimi due casi non saranno in seguito trattati perché la manipolazione delle stringhe non fa parte dei nostri compiti ed inoltre risulta di facile apprendimento laddove si sia compreso pienamente il funzionamento del pattern matching, termine che verrà chiarito in seguito.

27 Con il termine translitterazione si intende la sostituzione uno ad uno dei caratteri, ad esempio la regexp tr/A-Z/a-z/ trasforma tutti i caratteri maiuscoli in caratteri minuscoli.

3.2.1 Introduzione alle espressioni regolari

Le espressioni regolari non sono altro che sequenze di caratteri (spesso viste come una incomprensibile serie di simboli senza nessun senso) delimitate da un carattere di delimitazione, tipicamente lo slash (/); queste sequenze di caratteri sono comunemente chiamate pattern[28]: per vedere se un testo contiene il mio nome sarebbe sufficiente utilizzare come espressione regolare la stringa

```
/Tomaso/
```

Per applicare una espressione regolare ad un testo dovremmo usare l'operatore =~; ad esempio:

```
print “Ci sono!” if ($testo =~ /Tomaso/);
```

scrive “Ci sono!” solamente nel caso il mio nome sia presente nel testo contenuto nella variabile testo.

La tipologia di espressione da utilizzare, se ricerca, sostituzione o translitterazione, viene definita da una preposizione che può essere:

6. **m** per il pattern matching, ovvero il semplice riconoscimento del testo; questo è l'operatore di default quindi normalmente è sottinteso e viene inserito esclusivamente per motivi di leggibilità del codice
7. **s** per il pattern substituition, ovvero la sostituzione di porzioni di testo, ad esempio

   ```
   $testo=~s/Tomaso/Graziano/
   ```

   sostituirebbe Tomaso con Graziano nel contenuto della variabile testo
8. **tr** per la translitterazione; in questo caso il contenuto non è una

28 In inglese pattern significa letteralmente regola, ma più in generale il termine si riferisce al rispetto di una normativa, di una convenzione, ovvero ad uno schema: pattern venne tradotto, in ambito stilistico, come “cartamodello” per gli abiti; queste considerazioni vogliono solo giustificare l'utilizzo in seguito del termine inglese che risulterebbe troppo riduttivo nella traduzione “regola”.

vera e propria espressione regolare, bensì un semplice elenco di caratteri

9. Una espressione regolare può essere anche seguita da alcuni caratteri, detti modificatori, che permettono di influenzare le modalità di ricerca:

10. **i** indica che la ricerca deve essere fatta in modalità “case insensitive”, ovvero ignorando la differenza fra caratteri maiuscoli e minuscoli, ovvero `/ToMasO/i` e `/TomaSO/i` risultano in questo caso equivalenti

11. **m** indica di utilizzare separatamente ogni riga di testo, quindi gli operatori inizio e fine si riferiranno all'inizio ed alla fine di ogni riga piuttosto che del testo completo

12. **s** indica che l'operatore punto (.), che rappresenta qualsiasi carattere, e l'operatore spazio (\s) debbano considerare anche il carattere di nuova riga (\n)

13. **g** indica che la ricerca deve essere “globale”, ovvero non fermarsi alla prima occorrenza, bensì valutarle tutte, sia in fase di ricerca che di sostituzione, ovvero l'esempio `$testo=~s/Tomaso/Graziano/g` sostituirebbe ogni occorrenza di Tomaso invece che limitarsi alla prima.

Abbiamo visto sinora la “periferia” delle regexp, andiamo ora ad analizzarne il contenuto con un primo esempio di utilizzo:

```
print “Codice fiscale corretto” if ($cf =~ \
      /^[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]$/i)
```

la precedente l'espressione ci indica che la stringa cf deve iniziare (^) e finire ($) con una ben precisa sequenza di caratteri, ignorando se questi siano maiuscoli o minuscoli (appendice i); questa sequenza di caratteri deve essere composta da 6 caratteri alfabetici ([A-Z]), quindi da due caratteri numerici ([0-9]), poi una lettera, due numeri,

un'altra lettera, altri tre numeri ed infine una lettera; da questo esempio capiamo innanzitutto che esistono alcuni operatori per definire classi di caratteri ed altri per quantificare la loro numerosità.

Le classi di caratteri sono principalmente tre:

- **[...]** un carattere qualsiasi di quelli elencati, eventualmente elencati a gruppi come nell'esempio [123A-Za-p%$€]
- **[^...]** un carattere qualsiasi eccetto quelli elencati
- **.** qualsiasi carattere ad esclusione dell'a-capo (\n) a meno che non si utilizzi il modificatore s visto in precedenza

Per queste classi di caratteri esistono anche opportuni sinonimi, ad esempio **\s** rappresenta qualsiasi carattere di spaziatura, **\d** qualsiasi carattere numerico, ovvero equivale a [0123456789], o meglio alla sua sequenza contratta [0-9]; normalmente un sinonimo nella sua forma maiuscola rappresenta il contrario della sua forma minuscola, ad esempio **\D** equivale a [^0-9].

I quantificatori sono degli operatori che ci permettono di stabilire quante volte un carattere (o una sequenza) deve comparire:

- **{n}** la sequenza deve comparire esattamente n volte
- **{n,}** la sequenza deve comparire almeno n volte
- **{n,m}** la sequenza deve comparire almeno n volte e non più di m.
- * forma abbreviata di {0,}
- **+** forma abbreviata di {1,}
- **?** forma abbreviata di {0,1}

I quantificatori possono essere utilizzati anche nella forma detta “minimale” posponendo il quantificatore minimale caratterizzato dal punto interrogativo (?): approfondiremo il suo utilizzo fra poche ri-

ghe.

Un ultimo operatore essenziale per le espressioni regolari è l'operatore di raggruppamento, rappresentato dalle parentesi tonde, che permette di individuare specifiche aree del pattern da estrarre in seguito o manipolare in una operazione di sostituzione:

```
$cf = 'MNLTMS74M02L736Y';
$cf =~ /^([A-Z]{3})([A-Z]{3})(\d{2}[A-Z]\d{2})([A-Z]\d{3})([A-Z])/
print "$1 $2 $3 $4 $5";
```

PERL crea alcune variabili $1, $2, $3 e così via con i contenuti dei singoli gruppi racchiusi fra parentesi: l'esempio stampa il codice fiscale a blocchi, ovvero cognome, nome, data di nascita, luogo di nascita e carattere di controllo; si noti l'uso dell'espressione \d come sinonimo di [0-9].

La variabile $0 contiene, infine, il valore dell'intera espressione regolare incontrata: si pensi, nell'esempio precedente, ad escludere i caratteri di inizio e fine riga (^ e $), la variabile $0 conterrebbe il primo codice fiscale presente nel testo.

Riprendiamo ora l'operatore minimale per mostrarne un esempio, che risulterà molto utile nelle nostre analisi, poiché ci permette di chiudere la ricerca al primo raggiungimento della stessa, quindi di semplificare notevolmente la sintassi della nostra ricerca: vediamo come recuperare il contenuto della prima cella di una tabella, ignorando totalmente la struttura del contenuto della cella stessa:

```
$html =~ /<TABLE.*><TR.*><TD.*>(.*?)</TD>/i;
print $1;
```

se, nel caso precedente, avessimo omesso l'operatore minimale, la variabile $1 conterebbe tutto ciò che sta fra la prima cella della prima tabella e l'ultima cella dell'ultima tabella. Sfruttando questo esempio possiamo chiarire meglio il significato del modificatore g

nelle ricerche:

```
@celle = $html =~ /<TABLE.*><TR.*><TD.*>(.*?)</TD>/gi;
print @celle;
```

l'array @celle contiene il contenuto delle prime celle di ogni tabella presente nella pagina; notiamo anche l'uso del modificatore i, poiché l'HTML è un linguaggio non “case sensitive”; il modificatore g può essere inserito anche in un ciclo per effettuare specifiche operazioni ad ogni matching riscontrato:

```
my $i=0;
while ($html =~ /<TABLE.*><TR.*><TD.*>(.*?)</TD>/gi) {
      $i++;
      print “Contenuto della tabella $i: $1\n”;
}
```

3.3 Applicazione ai case study

Vediamo ora come applicare le espressioni regolari ai nostri problemi per estrarre i dati di nostro interesse: innanzitutto salviamo l'output degli script creati nel capitolo precedente in dei file html che analizzeremo per creare le regexp per l'estrazione dei contenuti informativi di nostro interesse:

```
perl CURL_polstrada.pl > polstrada.html
perl CURL_borsa.pl > borsa.html
perl MECHANIZE_arpa.pl > arpa.html
```

Aprendo i file con un editor notiamo che la zona della pagina dell'ARPA emiliana risulta pressoché illeggibile poiché la tabella contenente i dati di nostro interesse ci viene proposta in una unica riga: questo accorgimento è utilizzato da alcuni programmatori sia per motivi di performance (eliminando spazi, tabulazioni e caratteri di a-capo si riduce il traffico), sia per “offuscare” il codice, sia, infine, per semplice pigrizia; in casi come questi ci viene in aiuto Dave Ragget con il progetto tidy[29] che ci permette di indentare[30] correttamente il codice HTML: applichiamo la formattazione con il comando:

```
tidy -i arpa.html > arpa.tidy.html
```

Anche se il codice non è troppo offuscato è consigliabile applicare la formattazione per renderlo più leggibile. Dopo aver eseguito que-

29 http://tidy.sourceforge.net/

30 Con il termine indentare si intende la tecnica di isolare blocchi di codice in maniera grafica facendoli precedere da una tabulazione o da alcuni spazi, per evidenziare, ad esempio, il contenuto di una funzione, di un ciclo o di un tag HTML.

sta operazione il codice non sarà più identico al codice originario, ma con un opportuno utilizzo del gruppo di caratteri \s (spazio, tabulazione, ecc.), del modificatore s e dell'operatore minimale, si potrà applicare lo stesso pattern ai due file.

3.3.1 Dati Arpav

Vediamo innanzitutto come recuperare i valori del PM10[31] medio giornaliero nella zona della centralissima via Giardini, di via Nonantolana e del parco Ferrari: vediamo subito, dalla grafica che il dato si trova in una tabella in una cella a destra della sue definizione

Particolato < 10µm (PM10)		Media 24 ore (µg/m^3)	Superamenti VL (50 µg/m^3)	Superamenti consentiti	Confo DM60
Agglomerato R4 - MO	CARPI - CARPI 2 (VIA REMESINA)	35	39	35	
	MODENA - MO - PARCO FERRARI (PARCO FERRARI)	23	0		
	MODENA - MO - VIA GIARDINI (VIA GIARDINI)	38	54		No
	MODENA - MO - VIA NONANTOLANA (VIA CIMONE)	42	49		
Agglomerato R5 - MO	FIORANO MODENESE - SPEZZANO 2 (VIA MOLINO)	28	49	35	No

Il codice html da analizzare, riformattato con tidy, è il seguente:

```
[...]
   <td style="text-align: left;">
         MODENA - MO - PARCO FERRARI ( PARCO FERRARI )
   </td>
   <td>21</td>
   <td>0</td>
</tr>
[...]
```

31 Il PM10, più noto come polveri sottili, è tutto quel particolato con diametro inferiore ai 10 nanometri, particolarmente dannoso poiché traspira dagli alveoli al sangue: spesso il particolato è formato da metalli, anche pesanti, che si fissano alle cellule degradandole, nonché rendendo particolarmente difficile la respirazione; una seria politica di pulitura strade può incidere notevolmente su questo problema, poiché spesso il particolato che respiriamo è semplicemente quello depositatosi sollevato dal vento o dal passaggio delle autovetture.

Per estrarre i dati di parco Ferrari sarà quindi sufficiente la seguente espressione regolare:

```
$html =~ /MODENA \- MO \- PARCO FERRARI \( PARCO FERRARI \)<\/td><td>
(.*?)<\/td>/i;
$pm_10_ferrari = $1;
```

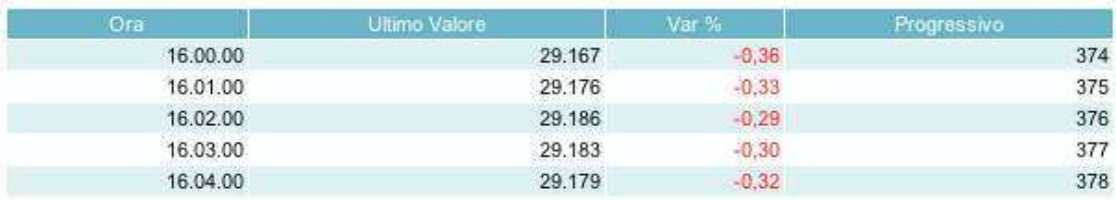
Notiamo l'uso dell'operatore minimale (?) per minimizzare il risultato: avremmo potuto farne a meno nel caso avessimo usato il gruppo di caratteri \d ([0-9]) che ci avrebbe vincolato a valori numerici, ma in questo caso ci saremmo ritrovati nel rischio di errore anche per piccole modifiche al codice della pagina, come l'inserimento di spazi o di numeri con decimali; come quantificatore si è utilizzato l'asterisco per comprendere anche i casi in cui non abbiamo nessun valore.

Utilizziamo dei pattern molto simili al precedente anche per via Giardini e per la Nonantolana:

```
$html =~ /MODENA \- MO \- PARCO FERRARI \( PARCO FERRARI \)<\/td><td>
(.*?)<\/td>/i;
$pm_10_ferrari = $1;
$html =~ /MODENA \- MO \- VIA GIARDINI \(VIA GIARDINI \)<\/td><td>
(.*?)<\/td>/i;
$pm_10_giardini = $1;
$html =~ /MODENA \- MO \- VIA NONANTOLANA \(VIA CIMONE \)<\/td><td>
(.*?)<\/td>/i;
$pm_10_nonantolana = $1;

print "\n\nParco Ferrari:   $pm_10_ferrari\n".
            "Via Giardini:    $pm_10_giardini\n".
            "Via Nonantolana: $pm_10_nonantolana\n";
```

3.3.2 Dati della Borsa italiana

Le operazioni da eseguire con il risultato della borsa italiana sono invero due: la prima operazione, assolutamente equivalente al caso precedente prevede di recuperare solamente il primo valore del risultato, mentre la seconda operazione prevede di recuperare tutti i dati presenti nella pagina, ovvero tutte le righe del risultato:



Ora	Ultimo Valore	Var %	Progressivo
16.00.00	29.167	-0,36	374
16.01.00	29.176	-0,33	375
16.02.00	29.186	-0,29	376
16.03.00	29.183	-0,30	377
16.04.00	29.179	-0,32	378

Nel caso in cui si voglia recuperare solamente la prima riga, usiamo come punto di partenza la sua intestazione con una espressione regolare un po' lunga ma sicuramente non troppo complessa: il codice html della sezione è il seguente:

```
<tr class="campi">
    <td>Ora</td>
    <td>Ultimo Valore</td>
    <td>Var %</td>
    <td>Progressivo</td>
</tr>
<!-- IF class=tdazz/tdbia -->
<tr class="tdazz">
    <td class="dato">10.30.00</td>
    <td class="dato">29.860</td>
        <!-- IF class=red/green/blu -->
    <td class="red">-0,22</td>
    <td class="dato">86</td>
</tr>
```

Vediamo ora l'espressione regolare da applicare: si noti che abbiamo inserito l'espressione regolare in una variabile per migliorare la leggibilità del codice.

```
$reg='<td>Ora<\/td>\s*'.
     '<td>Ultimo Valore<\/td>\s*'.
     '<td>Var \%<\/td>\s*'.
     '<td>Progressivo<\/td>\s*'.
     '<\/tr>.*?<tr class="tdazz">\s*'.
     '<td class="dato">(\d{2})\.(\d{2}).*?<\/td>\s*'.
     '<td class="dato">([0-9.,]*)<\/td>';

$html =~ /$reg/s;
$ore=$1;
$minuti=$2;
$dato=$3;
$dato_grezzo=$dato;
$dato =~ s/\.//;
$dato =~ s/\,/./;

print "\n\nOre $ore, Minuti $minuti, ".
      "Dato Grezzo $dato_grezzo, Dato $dato\n\n";
```

particolare attenzione va fatta per l'uso del modificatore s e dell'uso del quantificatore minimale per eliminare i commenti; un altro aspetto interessante da considerare è la sequenza delle trasformazioni da operare sul dato risultante per eliminare il punto di separazione delle migliaia e sostituire eventuali virgole, per le cifre decimali nel formato italiano, con punti, tipici del formato inglese: questa operazione risulta necessaria per poter trattare correttamente il dato numerico che la pagina ci fornisce in formato italiano ma che il nostro software deve gestire nel formato internazionale.

Poiché questa pagina in verità ci presenta cinque righe di dati, probabilmente il nostro desiderio è di recuperarle tutte e cinque in una volta: per fare questo utilizzeremo il modificatore **g** ed inseriremo l'espressione regolare in un ciclo; dobbiamo fare particolarmente attenzione alla sintassi della regular expression: poiché in questo caso, non potendo utilizzare il riferimento della riga di intestazione, dovremo definire più strettamente possibile la riga dei dati per evitare il rischio di ottenere dati non relativi alla zona di riferimento; ri-

sulta evidente che sarebbe decisamente più semplice operare eliminando dapprima la parte di testo non di nostro interesse, ma questo comporterebbe una complicazione nella successiva generalizzazione studiata ed esposta dell'ultimo capitolo, nonché, come detto in precedenza, per ridurre la quantità di codice da scrivere.

Nel codice seguente viene anche introdotto l'operatore OR (|): alcuni pattern racchiusi fra parentesi separati fra loro da questo operatore permettono di definire che il riconoscimento avvenga esclusivamente se uno solo di questi pattern compare in quella posizione, insomma il classico utilizzo dell'operatore OR logico:

```
my $reg='<\!\-\- IF class=tdazz\/tdbia \-\->\s*'.
       '<tr class\="(tdazz|tdbia)">\s*'.
          '<td class\="dato">(\d{2})\.(\d{2})\.00<\/td>\s*'.
          '<td class\="dato">([0-9.,]*)</td>\s*'.
          '<\!\-\- IF class=red\/green\/blu \-\->\s*'.
          '<td class\="(red|green|blu)">([0-9.,\-\+]*)<\/td>\s*'.
          '<td class\="dato">([0-9]*)<\/td>\s*'.
       '</tr>';

my @result;
my $i=0;

while ($html =~ /$reg/gs) {
      $result[$i]={
            'ore'=>$2,  'minuti'=>$3,
            'dato'=>$4, 'perc'=>$6, 'prog'=>$7
      };
      print "$result[$i]{ore}:$result[$i]{minuti}: ".
            "Dato $result[$i]{dato} - Prec $result[$i]{perc} - ".
            "Prog $result[$i]{prog}\n";
      $i++;
}
```

Il ciclo while si ripete finché non si è individuato l'ultimo pattern all'interno della stringa $html, ovvero finché l'espressione regolare non restituisca una “occorrenza non trovata”, ovvero false.

3.3.3 Dati della Polizia Stradale

L'esercizio per il sito della polizia stradale non si discosta di molto dai precedenti: i dati presenti nella pagina sono poco meno numerosi che nel caso dell'ARPA, ben formattati ed identificabili; in questo caso ci limiteremo, come esempio, ad estrarre semplicemente i dati delle pattuglie in servizio e degli incidenti rilevati, suddivisi fra strade ed autostrade.

		comunale		
Totale delle pattuglie impiegate *		595	831	1426
Attività infortunistica	Totale incidenti rilevati di cui:	87	135	222
	Incidenti mortali	0	6	6
	Incidenti con feriti	42	76	118

A differenza della pagina dell'ARPA, il codice risulta abbastanza indentato:

```
[...]
<tr>
                      <td colspan="3" class="testo12"><strong>Totale
delle pattuglie impiegate <span class="testorosso">*</span>
</strong></td>
                      <td align="center" class="tabellablu"><span
class="testo10">608</span> </td>
                      <td align="center" class="tabellablu"><span
class="testo10">822</span> </td>
                      <td align="center"
class="tabellarossacompleta"><span
class="testoRosso">1430</span> </td>
                    </tr>
                    <tr>
                      <td colspan="6" class="testo9"> </td>
                    </tr>
                    <tr>
                      <td width="140"
class="tabellarossacompleta">Attivit&agrave;<br>
                        infortunistica</td>
```

```
                      <td> </td>
                      <td> <div align="left"><span
class="testo10">Totale incidenti
                          rilevati<br>
                          di cui:</span></div></td>
                      <td align="center" class="tabellablu"><span
class="testo10">116</span>
                      </td>
                      <td align="center" class="tabellablu"><span
class="testo10">135</span></td>
                      <td align="center"
class="tabellarossacompleta"><span class="testoRosso">251</span></td>
                    </tr>
                    <tr>
                      <td> </td>
                      <td> </td>
                      <td> <div align="right"
class="testo10">Incidenti mortali</div></td>
                      <td align="center" class="tabellablu"><span
class="testo10">3</span></td>
                      <td align="center" class="tabellablu"><span
class="testo10">5</span></td>
                      <td align="center"
class="tabellarossacompleta"><span class="testoRosso">8</span></td>
                    </tr>
[...]
```

L'unica attenzione aggiuntiva da prestare in questo caso è che le righe non risultano separate solamente dal carattere \n, bensì dalla coppia \n\r, nello stile del mondo DOS/Windows: per questo motivo ci converrà utilizzare il modificatore s ed utilizzare la sequenza minimale per gli spazi, i tabulatori e gli a-capi fra i tag: `>.*?<`:

```
my $reg_cella_piena='<td.*?>.*?<span.*?>(\d*)<\/span>.*?<\/td>.*?';
my $reg_cella_vuota='<td.*?>.*?<\/td>.*?';

my $reg='Totale delle pattuglie impiegate.*?<\/td>.*?'.
       $reg_cella_piena.
       $reg_cella_piena;

$html =~ /$reg/s;
print "Pattuglie: autostrade $1, altre $2\n";

my $reg='Totale incidenti\s+rilevati.*?<\/td>.*?'.
       $reg_cella_piena.
       $reg_cella_piena;

$html =~ /$reg/s;
print "Incidenti: autostrade $1, altre $2\n";
```

4. Archiviazione dei Dati con Round Robin Database

Abbiamo completato con il capitolo precedente l'acquisizione dei dati da una sorgente web: in questo capitolo vedremo una particolare tecnica di archiviazione, ovvero le basi di dati “Round Robin”[32].

I dati, precedentemente acquisiti, tipicamente verranno archiviati in file di testo o in tabelle di database con le tecniche classiche; ciò ne permette una successiva analisi con strumenti informatici più opportuni, per eseguire operazioni di Datamining o altre analisi statistiche approfondite, quali delle cluster analsys o una analisi discriminanti per stabilire, ad esempio, gli effetti dell'applicazione della patente a punti sugli incidenti e sui decessi nelle strade italiane; lo scopo di queste pagine, però, è di integrare le informazioni ricevute con semplici grafici ed indici non esposti dalle sorgenti dati, generati in tempo reale non appena vengono aggiornati i dati nell'archivio.

Una caratteristica comune a tutti gli archivi presi sinora in considerazione è che si tratta sempre di serie temporali di dati: RRDTool è uno strumento disegnato appositamente per archiviare e generare report per questi tipi di dato.

Tobias Oetiker[33] ha , inizialmente, creato lo strumento dei Round Robin Database per archiviare e rendere in maniera grafica, dati ed indici di variazione del traffico di rete e dello stato di salute dei ser-

32 http://oss.oetiker.ch/rrdtool
33 System manager presso il Swiss federal institute of Technology

ver da lui gestiti[34]: questo strumento è stato poi generalizzato e viene utilizzato anche per dati quali il livello di fiumi, le temperature, ecc.

4.1 Il funzionamento di RRDTool

I database “Round Robin”, abbreviati in seguito con RRD, permettono di archiviare i dati in di file di dimensione fissa, mantenendo un puntatore allo stato corrente dei dati archiviati; si pensi, per chiarirsi le idee, ad un cerchio: ogni punto sulla circonferenza contiene un dato relativo ad un preciso istante; un vettore punta al valore corrente che, quando viene definito, si sposta di un passo lungo la circonferenza perciò dopo un po' di tempo questo vettore tornerà alla posizione iniziale, sovrascrivendo il dato inserito.

4.1.1 Differenze fra Round Robin ed i database

La prima paura che può scatenare questa tecnica in un normale utente di basi di dati è la perdita delle informazioni troppo vecchie:

34 Inizialmente, per archiviare i dati del suo programma di gestione della rete (MRTG), Toias aveva scelto di archiviare i dati in formato testuale ed aveva predisposto una serie di librerie in PERL per mantenere a dimensioni prefissate questi archivi; i dati archiviati erano, inoltre, esclusivamente numeri interi. Quando MRTG si è diffuso ed ha incominciato ad essere largamente utilizzato si è reso necessario un perfezionamento nell'archiviazione dei dati, per migliorare le performance e rendere lo strumento più flessibile: così queste librerie sono state riscritte in linguaggio C e vi è stato inserito anche il codice per la generazione dei grafici, rendendo così al contempo MRTG molto stabile e scalabile e rendendo autonomo RRDTool per essere utilizzato con qualsivoglia tipo di dati.

questa paura, benché reale, è, però, nel nostro caso, un problema di poco conto: utilizzeremo, infatti, questo strumento solamente per creare delle semplici analisi su periodi prefissati, mentre i dati storici verranno archiviati in maniera classica in dataset su file di testo[35].

Ma se archiviamo i dati anche in maniera classica perché utilizzare questo strumento? Questo strumento non serve semplicemente ad archiviare i dati, bensì si presta perfettamente per la creazione di semplici statistiche e diagrammi delle serie storiche acquisite, indicandoci anche se qualche dato risulta macante.

RRD non è un motore di database, bensì è uno strumento di archiviazione dati ed al contempo uno strumento per la loro presentazione; un altra peculiarità è che, non crescendo la dimensione degli archivi, questo strumento non richiede grosse risorse e soprattutto elimina qualsiasi studio di scalabilità nelle applicazioni da lui derivate.

I database classici archiviano normalmente i dati puri che gli vengono forniti, mentre RRD può archiviare anche solamente la differenza fra i vari stati; si pensi all'archiviazione dei dati del livello della marea: fissato lo zero mareografico, il nostro interesse è di sapere quale movimento ha avuto la marea stessa, ovvero il differenziale, delta, fra un istante ed il successivo; per creare grafici sull'andamento delle maree e sui picchi di variazione non ci interessa il valore assoluto, ma solamente la variazione fra un'ora e l'altra: archiviare solo questo dato ci permette di ottenere esattamente i grafici e gli indici che ci interessano, diminuendo notevolmente la dimensione dell'archivio che li conterrà e riducendo il numero di conti da effettuare per le semplici analisi richieste.

Una ultima peculiarità di RRD è che, essendo espressamente stu-

35 Si ribadisce che Rond Robin Database non è un sostituto dei Database classici, bensì un sistema di archiviazione altamente specializzato per i dati derivanti da serie storiche.

diato per l'archiviazione di serie storiche, ogni valore nell'intervallo di tempo definito, se non fornito verrà archiviato come sconosciuto, per evitare di presentare interpolazioni non volute: le basi di dati classiche non distinguono veramente le serie storiche da altri tipi di archivio, perciò deve essere il data base designer a strutturare di volta in volta lo schema ed il software per l'archiviazione dei dati per prevedere la presenza di dati mancanti.

4.1.2 Utilizzo e funzionalità di RRDTool

In questa sezione vedremo come utilizzare, dalla linea di comando della shell, RRDtool; l'inserimento dei dati nei nostri case study avverrà, invece, con PERL, ma i concetti, come vedremo, resteranno comunque gli stessi.

Come primo esempio vediamo di creare un archivio che contenga il numero di incidenti stradali giornalieri dal primo gennaio 2002 per cinque anni consecutivi:

```
rrdtool create incidenti_stradali.rrd \
      --start 1009836000 \
      --step 86400 \
      DS:incidenti:GAUGE:172800:0:10000 \
      RRA:AVERAGE:0.5:1:1825
```

L'unica parte auto esplicativa di questa istruzione è la creazione dell'archivio incidenti_stradali.rrd, mentre il resto risulterà al lettore abbastanza misterioso; la data di inizio dell'archivio, cosiccome tutte le date in seguito utilizzate da questo tool, sono espresse nel numero

di secondi dal primo gennaio 1970 (data definita epoch in inglese): 1009836000 è il numero di secondi trascorsi dal primo gennaio 1970 al primo marzo del 2001; chiarito che RRDTool sfrutta i secondi, risulta più chiaro il significato del passo che, essendo espresso in secondi, indica un giorno intero, ovvero 60 secondi moltiplicati per 1440 minuti, ovvero 24 ore.

Per rendere più chiara la sintassi dei nostri comandi utilizzeremo l'operatore backtik (`) che ci permette di inserire in una linea di comando l'output di un altro comando: sfruttiamo quindi il comando bc e lo script date2ts (il cui codice si può trovare in appendice); bc permette di eseguire a linea di comando semplici operazioni aritmetiche, ad esempio:

```
echo 60*60*24 | bc
```

stampa il risultato 86400; date2ts trasforma una data in formato discorsivo nel numero di secondi trascorsi dal primo gennaio 1970:

```
date2ts 01/01/2001 #stampa 1009836000
date2ts oggi #valore alla mezzanotte di oggi
date2ts adesso #valore all'istante
date2ts ieri
```

Riscrivendo l'esempio precedente utilizzando questi strumenti risulterà più facile al neofita, ma anche al veterano, comprende il significato delle prime tre righe:

```
rrdtool create incidenti_stradali.rrd \
      --start `./date2ts 01/01/1970` \
      --step `echo 60*60*24 | bc` \
      DS:incidenti:GAUGE:172800:0:10000 \
      RRA:AVERAGE:0.5:1:1825
```

Nelle ultime due righe viene definito il dato vero e proprio ed il sistema di archiviazione dello stesso; il dato viene definito nella forma:

```
DS:nome variabile:DST:heartbit:min:max
```

dove DST (Data Source Type) indica il tipo di dato che viene forni-

to tra i quattro disponibili:

- COUNTER: la variabile è un contatore, ovvero un numero che incrementa sempre; per questo motivo RRD archivierà il valore iniziale e in seguito solo gli scostamenti da questo; si pensi ad esempio al numero di chilometri totali percorsi da una automobile
- DERIVE: la variabile è un contatore che ammette anche incrementi negativi; anche in questo caso viene memorizzato solo lo scostamento dal valore precedente; si pensi ad esempio al valore della marea o al valore assoluto dell'indice di borsa
- ABSOLUTE: la variabile è un contatore, ma ci viene fornita solo la differenza dallo stato precedente; riallacciandosi al primo esempio del chilometraggio dell'automobile, in questo caso vengono passate esclusivamente le distanze parziali
- GAUGE: la variabile è un valore assoluto: non viene quindi salvata la differenza dal valore precedente, bensì il valore vero e proprio.

Di seguito è definito l'"heartbeat"[36] della variabile ed eventualmente il suo valore minimo e massimo; se si cerca di impostare un valore al di fuori dell'intervallo definito RRD imposterà automaticamente ad UNKNOWN il valore, trattandosi di un dato non considerato reale: sarà eventualmente il parametro "heartbeat" a valorizzare l'area, se necessario, con la media dei valori limitrofi.

Dopo aver definito le variabili in gioco, per ogni variabile possiamo definire il metodo di archiviazione con la definizione del RRA (round robin archive):

```
RRA:CF:xff:step:rows
```

dove CF è la funzione di consolidamento, ovvero AVERAGE (me-

36 Si veda l'appendice "Hearbeat e RRD" per una descrizione dettagliata di questo parametro.

dia), MINUIMUM (minimo), MAXIMUM (massimo) o LAST (ultimo valore inserito); xff ci indica quanti dati mancanti rendono nulla la funzione, ovvero oltre quale percentuale di dati certi la funzione ritorna un valore consistente.

Il parametro steps indica quanti valori devono essere considerati dalla funzione: utilizzarne solo uno, come nel nostro esempio, significa avere la media del solo ultimo valore, ovvero il valore stesso.

Il parametro rows, infine, indica quanti dati di questa funzione saranno archiviati, ovvero dopo quante scritture verrà perso il primo valore calcolato: nel nostro esempio, poiché calcoliamo un valore al giorno, 1825 (365*5) indica che vogliamo mantenere i dati degli ultimi cinque anni.

Per ogni variabile si può inserire anche più di una archivio: ad esempio può essere interessante mantenere l'andamento dell'ultimo mese e dell'ultimo anno, costruendo la media settimanale e mensile rispettivamente:

```
rrdtool create incidenti_stradali.rrd \
      --start `./date2ts 01/01/1970` \
      --step `echo 60*60*24 | bc` \
      DS:incidenti:GAUGE:`echo 60*60*24*2 | bc`:0:10000 \
      RRA:AVERAGE:0.5:1:`echo 365*5 | bc`
      RRA:AVERAGE:0.5:7:4
      RRA:AVERAGE:0.5:30:12
```

Ora che abbiamo creato il nostro archivio, vediamo come inserirvi i dati da linea di comando:

```
rrdtool update incidenti_stradali.rrd `./date2ts 01/01/2002`:259
rrdtool update incidenti_stradali.rrd `./date2ts 01/01/2003`:264
rrdtool update incidenti_stradali.rrd `./date2ts 01/01/2004`:285
rrdtool update incidenti_stradali.rrd `./date2ts 01/01/2005`:229
```

e quindi come estrarli:

```
rrdtool fetch incidenti_stradali.rrd AVERAGE \
      --start `./date2ts 01/01/2002` \
      --end `./date2ts 03/01/2002`
```

il risultato dell'estrazione sarà preceduto dalla data nel formato tipico di RRD del numero dei secondi dal 1970, ovvero:

```
1009843200: 1.0508333333e+02
1009929600: 1.3508333333e+02
1010016000: 9.2750000000e+01
```

Per rendere leggibile questo elenco, utilizziamo un altro piccolo script, rrd2human (che troveremo discusso in appendice), che trasforma questo output in un formato più leggibile:

```
rrdtool fetch incidenti_stradali.rrd AVERAGE \
      --start `./date2ts 01/01/2002` \
      --end `./date2ts 03/01/2002` |
      rrd2human
```

che produrrà come output:

```
0001  1/01/2002 02:00:00 -->      105
0002  2/01/2002 02:00:00 -->      135
0003  3/01/2002 02:00:00 -->       92
```

4.2 Utilizzo di RRDtool con i dati polstrada

Per non annoiare ulteriormente il lettore, ci limiteremo ad esporre come archiviare i dati della Polstrada, lasciando comunque la possibilità di leggere in appendice le impostazioni utilizzate per gli altri due case study.

Dalla pagina informativa della Polstrada ci limiteremo ad acquisire i dati sugli incidenti e sulle pattuglie presenti, trascurando la sezione delle contravvenzioni, dell'attività di soccorso e sui provvedimenti sanzionatori.

Totale delle pattuglie impiegate *		628	761	1389
Attività infortunistica	Totale incidenti rilevati di cui:	94	123	217
	Incidenti mortali	0	1	1
	Incidenti con feriti	18	57	75
	Incidenti con danni	76	65	141
	Persone decedute	0	1	1
	Persone ferite	27	77	104

Vediamo quindi che entrano in gioco le seguenti sorgenti di dati: le pattuglie impiegate (inserite nella variabile pattuglie), gli incidenti totali (incidenti), quelli mortali (imortali), quelli con feriti (iferiti), le persone decedute (deceduti) e quelle ferite (feriti); il dato degli incidenti con danni verrà trascurato in quanto sarà possibile ricavarlo per differenza; ogni variabile avrà due suffissi: ss per le strade statali e as per le autostrade.

Salta immediatamente all'occhio che lo “step” per i nostri dati è

giornaliero; la data iniziale è il primo marzo del 2001 (primo giorno disponibile negli archivi); la variabile pattuglie non sarà disponibile fino al primo settembre 2004, risulterà perciò fino a questa data pari a zero.

Ogni variabile presa in considerazione avrà un minimo pari a zero ed un massimo indefinito, ma che possiamo impostare indicativamente a 1.000; l'heartbeat della variabile sarà pari a due giorni (per imporci al massimo un giorno di dati mancanti); il tipo di Data Source è indubbiamente di tipo GAUGE, poiché non stiamo in nessun caso parlando di contatori. Si potrebbe considerare, in verità, il contatore dei morti e degli incidenti nel periodo, quindi considerare le variabili di tipo ABSOLUTE, poiché il dato fornitoci è pari alla differenza dal giorno precedente dopo aver azzerato il contatore, ma personalmente faccio un po' di difficoltà a considerare le vite umane come un numero, perciò, vista l'ininfluenza nella nostra trattazione, ho deciso di optare per il tipo GAUGE.

Per ogni variabile ci può interessare innanzitutto l'archiviazione del suo valore per almeno dieci anni, per poi avere dei grafici più sintetici con l'andamento nelle ultime quattro settimane, negli ultimi tre mesi, negli ultimi dodici mesi e negli ultimi tre anni: questa considerazione ci impone di creare cinque archivi:

- il dato stesso archiviato per 3650 volte
- il dato stesso archiviato per 28 volte
- il dato stesso archiviato per 90 volte
- la media di sette dati archiviata per 52 volte
- la media di sette dati archiviata per 150 volte

La media viene fatta al più settimanalmente per evitare di appiattire eccessivamente i grafici, col rischio di renderli non significativi.

Queste operazioni impongono l'archiviazione quotidiana dei dati del giorno precedente e la creazione di un archivio polstrada.rrd con i seguenti parametri:

```
rrdtool create polstrada.rrd \
      --step `echo 60*60*24 | bc` \
      --start `./date2ts 28/02/2001` \
      DS:pattuglie_as:GAUGE:172800:0:5000 \
      DS:incidenti_as:GAUGE:172800:0:1000 \
      DS:imortali_as:GAUGE:172800:0:1000 \
      DS:iferiti_as:GAUGE:172800:0:1000 \
      DS:deceduti_as:GAUGE:172800:0:1000 \
      DS:feriti_as:GAUGE:172800:0:1000 \
      DS:pattuglie_ss:GAUGE:172800:0:5000 \
      DS:incidenti_ss:GAUGE:172800:0:1000 \
      DS:imortali_ss:GAUGE:172800:0:1000 \
      DS:iferiti_ss:GAUGE:172800:0:1000 \
      DS:deceduti_ss:GAUGE:172800:0:1000 \
      DS:feriti_ss:GAUGE:172800:0:1000 \
      RRA:AVERAGE:0.5:1:3650 \
      RRA:AVERAGE:0.5:1:28 \
      RRA:AVERAGE:0.5:1:90 \
      RRA:AVERAGE:0.5:7:52 \
      RRA:AVERAGE:0.5:7:150
```

L'archivio così creato ha una dimensione inferiore ai 400 Kb e questa dimensione non varierà mai: è strabiliante notare come RRD sia uno strumento decisamente poco affamato di risorse!

Risulta interessante notare da questo esempio come, benché i “data source” siano numerosi, non occorre definire il tipo di archivio per ogni singolo dato, bensì il metodo di archiviazione sarà omogeneo per ognuno di essi; ecco finalmente un punto in comune con i database classici, ovvero che un archivio dati, una tabella, secondo i dettami classici, è bene che contenga solamente informazioni coerenti fra loro stesse, per evitare che gli archivi creati risultino disomogenei oppure non significativi; il principio degli archivi RRD è di contenere serie storiche: poiché su queste serie storiche vengono effettuate delle operazioni aritmetiche è bene evitare di inserire dati qualita-

tivi per evitare, ad esempio, di avere la media del sesso dei passanti o la media del loro colore i capelli.

Per inserire i dati attraverso PERL nell'archivio così creato utilizzeremo la libreria RRDTool::OO, che ci permette di utilizzare una struttura “hashref”[37] per aggiornare le singole variabili:

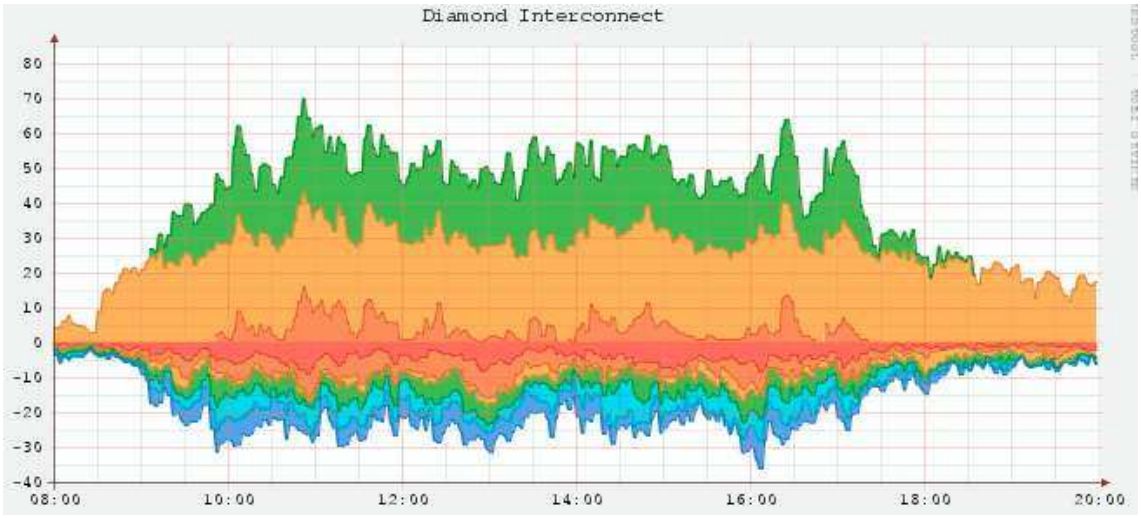
```
use RRDTool::OO;

my $GG='05';
my $MM='05';
my $AAAA='2006';

[...]

my $rrd = RRDTool::OO->new(
        file => "polstrada.rrd" );

my %dati;

[...]

my $reg='Totale delle pattuglie impiegate.*?<\/td>.*?'.
         $reg_cella_piena.
         $reg_cella_piena;
$html =~ /$reg/s;
$dati{'pattuglie_as'}=$1;
$dati{'pattuglie_ss'}=$2;

my $reg='Totale incidenti\s+rilevati.*?<\/td>.*?'.
         $reg_cella_piena.
         $reg_cella_piena;
$html =~ /$reg/s;
$dati{'incidenti_as'}=$1;
$dati{'incidenti_ss'}=$2;

&Date_Init("Language=Italian","DateFormat=non-US");
my $date = ParseDate("$GG/$MM/$AAAA");

$rrd->update(time => UnixDate($date, "%s")+7200,
        values => \%dati);
```

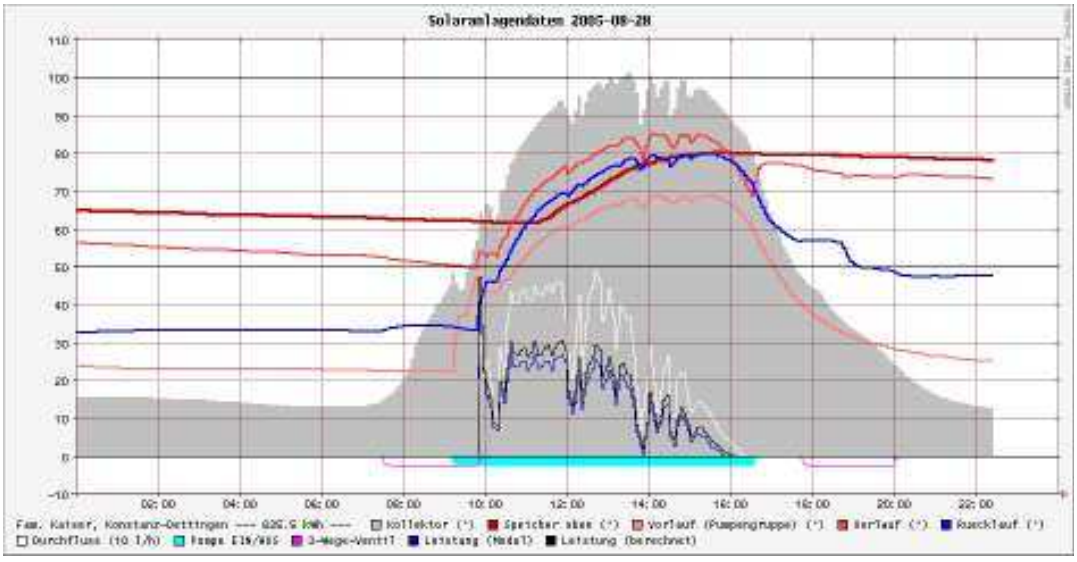
37 Con il termine hash vengono definiti in PERL gli array associativi, ovvero quei vettori di dati il cui record non è individuato da un numero ma da una chiave testuale. Il termine hasref indica il puntatore ad una struttura di tipo hash.

Si è optato per questa libreria per la semplicità di utilizzo e perché ci permette di utilizzare un hashref per l'inserimento dati: non utilizzando il nome della variabile nella definizione della chiamata alla funzione, ma semplicemente nella definizione degli elementi della variabile otterremo una maggiore flessibilità per la generalizzazione dell'operazione ed una più semplice operazione di debug degli script; per la creazione degli archivi e dei grafici si può adottare la più nota RRD::Simple che permette di utilizzare una sintassi più vicina a quella dello strumento a riga di comando e che è, fra l'altro, normalmente già presente nelle distribuzioni PERL; noi, nel seguito, preferiamo utilizzare direttamente la definizione del comando RRD per permettere una parametrizzazione degli script più chiara e meno ridondante di parametri.

5. Grafici con RRDtool

Una delle principali caratteristiche di RRDTool è la possibilità di creare report dai propri archivi. Per creare i grafici RRD acquisisce i dati dai propri archivi con il metodo fetch (intravisto nel precedente capitolo) quindi, basandosi su diverse opzioni, crea dei grafici delle serie temporali ottenute.





5.1 Cenni sul fetch dei dati

Poiché la creazione dei grafici è strettamente legata all'acquisizione dei dati, prima di proseguire nei dettagli delle opzioni disponibili, vediamo più in dettaglio come acquisire dati da un archivio RRD:

```
rrdtool fetch polstrada.rrd AVREAGE \
      --resolution `echo 60*60*24 | bc` \
      --start `./date2ts 01/01/2005` \
      --end `./date2ts 31/12/2005`
```

l'acquisizione dei dati da un archivio prevede innanzitutto di definire l'intervallo di tempo desiderato (--start e --end), ed una “risoluzione” (--resolution) su questo intervallo di tempo: questo ci fa pensare che nel capitolo precedente è stato commesso un errore nel creare degli archivi che si comprendono, come intervallo, l'uno con l'altro; questo non è un errore, bensì un eccesso di prudenza, poiché ci permette di evitare che l'operazione di fetch esegua delle interpolazioni e dei conti non previsti; con grosse quantità di dati, inoltre, se non viene specificato un opportuno archivio, l'operazione di fetch potrebbe impiegare una considerevole quantità di risorse di sistema (memoria e processore) nonché di tempo.

Si noti infine che l'operazione di fetch ci permette, anzi ci impone, di definire una funzione cumulata (definita CF nella documentazione) da applicare ai dati: a differenza degli altri parametri la definizione della funzione è l'unico parametro obbligatorio perché indica ad RRD a quale archivio accedere, mentre l'intervallo di tempo e la risoluzione, se non specificate, indicano di acquisire solamente i dati del giorno precedente con la massima risoluzione disponibile.

5.2 Creazione di grafici

RRDTool ci permette di creare grafici che contengono anche più sorgenti di dati contemporaneamente e, addirittura, ci permette di acquisire sorgenti dati da diversi archivi: potremmo, ad esempio, creare grafici che contengano contemporaneamente informazioni sugli incidenti stradali e sull'inquinamento: questo dettaglio rende palese la possibilità di utilizzare RRD anche come uno strumento di base per effettuare analisi di tipo data warehouse, ovvero di confrontare archivi eterogenei in un unico grafico.

Un altra interessante potenzialità della generazione di grafici con RRDTool, è che sulle variabili possono essere effettuate delle operazioni prima della loro visualizzazione, usando lo standard delle espressioni aritmetiche RPN, derivato da quello utilizzato nelle calcolatrici scientifiche della HP.

Infine, in un grafico è possibile inserire anche dei dati di sintesi sulle variabili e quindi, paradossalmente, un grafico potrebbe essere anche un report numerico sullo stato di alcune variabili: questo aspetto spiega l'intercambiabilità del termine grafico e report in queste righe.

La sintassi base per la creazione dei grafici è la seguente:

```
rrdtool graph graphic.png \
      [option ...] \
      [data definition ...] \
            [data calculation ...] \
            [variable definition ...] \
      [graph element ...] \
            [print element ...]
```

Andiamo a scoprire ora il significato e le potenzialità di ogni singola sezione.

5.2.1 Rrdtool graph: opzioni

La creazione di un report prevede innanzitutto di definire come e dove sarà creato il grafico: i formati possibili sono png (bitmap), svg o eps (vettoriali) e dipendono direttamente dall'estensione del file (nell'esempio precedente graphic.png); si può inviare il grafico direttamente in output utilizzando la keyword – (segno meno).

Per definire l'intervallo temporale del grafico si utilizzano le opzioni start e end, specificando eventualmente il passo (step) da utilizzare; si possono utilizzare le macro[38] end o start per definire un intervallo di tempo relativo, ad esempio:

```
--end now -start end-`echo 60*60*24*30|bc` --step 86400
```

creerà un grafico degli ultimi trenta giorni con passo di un giorno (86400 secondi).

Si possono usare le opzioni title e vertical-label per inserire un titolo o una etichetta per le ordinate del grafico; il risultato sarà normalmente dimensionato in 400 pixel orizzontali e 100 verticali, a meno di non valorizzare le opzioni width ed height: può anche essere creata una thumbnail[39] del grafico utilizzando una altezza inferiore ai 32 pixel combinata con il parametro only-graph.

Per definire la risoluzione del diagramma temporale si possono stabilire i limiti massimi e minimi (upper e lower-limit), che saranno comunque ignorati nel caso di superamento a meno di non indicare l'opzione rigid, che vincolerà rigidamente la dimensione delle ordina-

38 Con il termine macro si indicano le macroistruzioni, ovvero termini che vengono sostituiti nel codice prima della loro analisi: nel nostro esempio start indica il valore passato all'omonimo parametro; il discorso è analogo con la macro end.

39 Letteralmente “unghia del pollice”, thumbnail indica quelle immagini visualizzate come anteprima nelle gallerie fotografiche, comparabili con i provini a cui siamo abituati dal fotografo

te ai parametri di limite specificati; i colori utilizzati possono essere definiti puntualmente, nel formato esadecimale rosso, verde e blu per ogni parte del grafico: sfondo (back), sfondo del grafico (canavas), ombre dei bordi(shadea/b), griglia (grid), testo (font) e assi (axis); i colori delle linee temporali saranno definiti direttamente nella definizione della stessa, come si leggerà in seguito.

Un'altra caratteristica interessante è la possibilità di inserire un watermark nel grafico e di modificare i parametri di scala degli assi; infine è possibile definire la suddivisione delle ascisse e delle ordinate agendo sui parametri x-grid e y-grid.

5.2.2 Rrdtool graph: definizione dati, variabili e dati calcolati

In questa sezione vedremo come è possibile definire quali informazioni disegnare: possiamo inserire in un grafico i dati di un archivio (data definition: DEF), delle variabili calcolate (variable definition: VDEF) o dei calcoli su gruppi di dati (data calculation: CDEF).

Vediamo innanzitutto come acquisire i dati da un archivio:

```
DEF:variabile=archivio.rrd:datasource:CF: \
      step=passo:start=inizio:end=fine:reduce=CF
```

vediamo che per ogni variabile possiamo utilizzare un archivio RRD differente, da ogni archivio possiamo utilizzare un singolo datasource, definendo la sua funzione di accumulo (CF) e l'intervallo di dati di interesse, esattamente con le stesse notazioni già viste nel paragrafo precedente; si può eventualmente applicare una seconda funzione di accumulo (reduce) da utilizzare nel momento in cui i dati devono essere adattati alla dimensione del grafico.

Si può definire anche una variabile scalare, opzione molto utile nella creazione di report numerici o per un suo utilizzo all'interno delle definizioni di dati calcolati che vedremo in seguito:

```
VDEF:variable=RPN expression
```

dove “RPN expression” indica una espressione aritmetica scritta usando la sintassi tipica delle calcolatrici scientifiche HP, ad esempio:

```
VDEF:massimo=dati,MAX
```

calcola il massimo della sorgente dati e lo salva nella variabile massimo; possono essere utilizzati anche operatori condizionali:

```
VDEF:buono=massimo,10,LT,0,1,IF
```

che assegna alla variabile buono il valore 1 se massimo è maggiore di 10, 0 altrimenti; l'operazione va letta da sinistra a destra per blocchi funzionali, quindi innanzitutto viene valutata l'espressione massimo,10,LT, che equivale a massimo<10 nella sintassi convenzionale, e quindi viene calcolata l'espressione ternaria a,b,c,IF che equivale, nella sintassi PERL a scrivere:

```
if (a) {
      return b;
} else {
      return c;
}
```

questo meta linguaggio ci permette di effettuare anche alcuni utili calcoli statistici, quali il percentile (PERCENT), la media (AVERAGE) e una analisi di regressione dei minimi quadrati calcolando l'intercetta (LSLINT), il parametro angolare (LSLSLOPE) ed il coefficiente di correlazione di Pearson (LSLCORREL)

Dopo aver acquisito i dati e creato delle variabili (scalari), può interessarci creare un ulteriore gruppo di dati da rendere in maniera

grafica:

```
CDEF:array=PRPN expression
```

A differenza della definizione VDEF precedente, in questo caso viene creato un nuovo set di dati, ad esempio per combinare insieme due sorgenti dati acquisite con la clausola DEF; per rendere più chiaro il significato di questa definizione pensiamo ai dati acquisiti dal sito della polizia di stato: abbiamo visto nel precedente capitolo che archiviamo in due dataset diversi gli incidenti nelle autostrade e quelli nelle strade statali: se noi volessimo creare un grafico con gli incidenti totali, in strade ed autostrade dovremmo scrivere:

```
DEF:incidenti_as=polstrada.rrd:incidenti_as:AVERAGE
DEF:incidenti_ss=polstrada.rrd:incidenti_ss:AVERAGE
CDEF:incidenti_totali=incidenti_as,incidenti_ss,+
```

Vediamo un ulteriore esempio che mostra come utilizzare le variabili all'interno di dati calcolati: pensiamo di creare un dataset che rappresenti gli scostamenti dalla media dei nostri dati:

```
DEF:incidenti_as=polstrada.rrd:incidenti_as:AVERAGE
VDEF:media=incidenti,AVERAGE
CDEF:scostamenti=incidenti,media,-
```

La creazione di espressioni RPN può risultare, in alcuni casi, molto complessa, soprattutto quando dobbiamo effettuare operazioni raggruppate, ad esempio (a+b)*c: per evitare problemi e semplificare la lettura del codice prodotto si consiglia il lettore di sfruttare eventuali variabili di supporto per isolare le singole operazioni:

```
VDEF:abc=a,b,+,c*
#oppure
VDEF:ab=a,b,+
VDEF:abc=ab,c*
```

5.2.3 Rrdtool graph: elementi di report e grafici

In ogni immagine generata con rrdtool si possono inserire vari elementi grafici (linee, movimenti, aree) e testuali: in generale viene creato un report che, in presenza di elementi grafici, esporrà nella parte superiore un diagramma delle sorgenti dati in gioco.

Per stampare il valore di una variabile, tipicamente definita in una espressione VDEF, si utilizza il comando PRINT, nel quale viene definita anche la formattazione da utilizzare:

```
PRINT:percentuale:La percentuale è del %2.1lf%%
```

Se si volesse inserire del testo all'interno dell'area del grafico, dopo la legenda sottostante al grafico, si utilizza, invece, GPRINT, con la medesima sintassi; per inserire in questa stessa area del testo libero si utilizza il comando COMMENT.

La possibilità più interessante è di rendere graficamente l'andamento di una variabile temporale, definita con DEF o CDEF: il comando LINE inserisce questo elemento grafico:

```
LINE:variabile:#ccddee:leggenda
```

per ogni variabile viene definito un colore ed una legenda; opzionalmente si può inserire il comando STACK al termine della definizione per far si che la nuova linea si “appoggi” sulla linea precedente, ovvero per dare una rappresentazione grafica del df(x): ad esempio se noi volessimo riportare in un grafico gli incidenti nelle autostrade e quelli totali, per proseguire il precedente esempio, potremmo utilizzare equivalentemente la forma:

```
LINE:incidenti_as:#0000ff:Incidenti nelle autostrade
LINE:incidenti_totali:#ff0000:Incidenti totali
```

oppure,eliminando l'utilizzo della CDEF:

```
LINE:incidenti_as:#0000ff:Incidenti nelle autostrade
LINE:incidenti_ss:#ff0000:Incidenti totali:STACK
```

Per marcare particolari momenti temporali nel grafico può risultare molto utile inserire delle linee verticali di demarcazione utilizzando la funzione VRULE.

Una alternativa al metodo LINE è il metodo AREA che permette di colorare tutta l'area compresa fra l'asse temporale, l'ascisse, e la linea dell'andamento.

Per confrontare dati in tempi diversi è indispensabile “attualizzare” la posizione temporale dei dati: ad esempio per paragonare i dati di borsa odierni con i dati di ieri, bisogna far si che i dati di di ieri siano spostati di 24 ore, ovvero di 86400 secondi; per fare ciò usiamo il comando SHIFT:

```
SHIFT:dati_di_ieri:86400
```

5.3 Esempio di grafici per i dati della polstrada

Cerchiamo di chiarire le idee sulla creazione dei grafici creando un grafico utilizzando i dati della polstrada per confrontare l'andamento degli incidenti nel quadriennio dal 2002 al 2005 per avere una visione grafica che ci permetta di capire se ci sono stati cambiamenti sostanziali di comportamento fra questi anni. Per fare ciò dobbiamo acquisire i dati dallo stesso archivio e dalla stessa sorgente dati (ad esempio incidenti_ss, ma con tre intervalli temporali diversi:

```
DEF:incidenti2002=polstrada.rrd:incidenti_ss:AVERAGE: \
      end=`date2ts 01/01/2003`:start=end-1y
DEF:incidenti2003=polstrada.rrd:incidenti_ss:AVERAGE: \
      end=`date2ts 01/01/2004`:start=end-1y
DEF:incidenti2004=polstrada.rrd:incidenti_ss:AVERAGE: \
      end=`date2ts 01/01/2005`:start=end-1y
DEF:incidenti2005=polstrada.rrd:incidenti_ss:AVERAGE: \
      end=`date2ts 01/01/2006`:start=end-1y
```

Acquisiti i dati dovremmo riposizionare i dati precedenti al 2005 spostandoli di uno, due e tre anni rispettivamente:

```
SHIFT:incidenti2004:`echo 60*60*24*365*1 | bc`
SHIFT:incidenti2003:`echo 60*60*24*365*2 | bc`
SHIFT:incidenti2002:`echo 60*60*24*365*3 | bc`
```

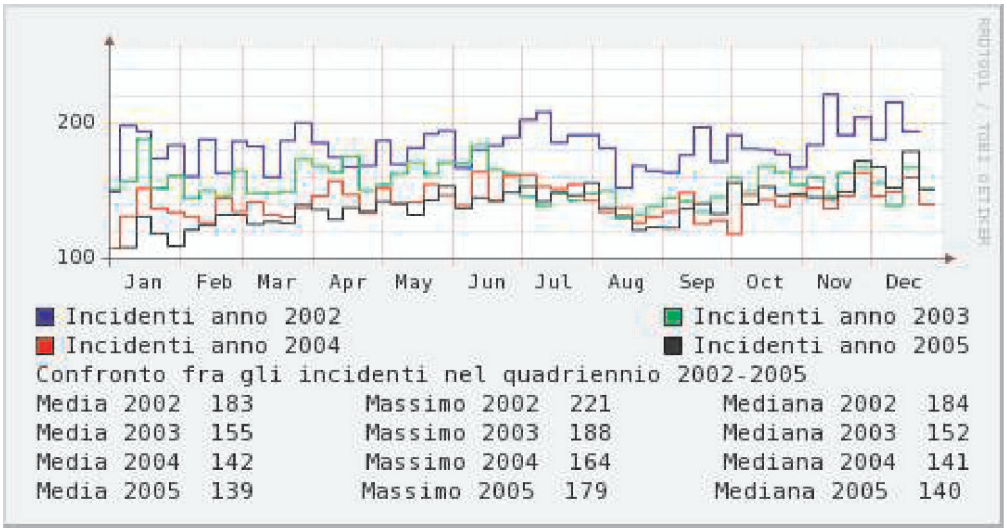
Ora non resta che calibrare su un anno la dimensione del nostro grafico (il 2005) e definire a livello generale la risoluzione da adottare (step), nel nostro caso settimanale: la risoluzione giornaliera ci fornirebbe un grafico troppo fitto, quasi illeggibile, perciò adottando un passo più lasco riusciremo a migliorarne la lettura.

```
--end `./date2ts 01/01/2006` \
--start end-1y \
--step `echo 60*60*24*7|bc`
```

Mettendo tutto assieme, otteniamo come comando per la generazione del grafico:

```
rrdtool graph polstrada.png \
      --end `./date2ts 01/01/2006` \
      --start end-1y --step `echo 60*60*24*7|bc` \
      DEF:incidenti2002=polstrada.rrd:incidenti_ss:AVERAGE: \
            end=`./date2ts 01/01/2003`:start=end-1y \
      DEF:incidenti2003=polstrada.rrd:incidenti_ss:AVERAGE: \
            end=`./date2ts 01/01/2004`:start=end-1y \
      DEF:incidenti2004=polstrada.rrd:incidenti_ss:AVERAGE: \
            end=`./date2ts 01/01/2005`:start=end-1y \
      DEF:incidenti2005=polstrada.rrd:incidenti_ss:AVERAGE: \
            end=`./date2ts 01/01/2006`:start=end-1y \
      SHIFT:incidenti2004:`echo 60*60*24*365   | bc` \
      SHIFT:incidenti2003:`echo 60*60*24*365*2 | bc` \
      SHIFT:incidenti2002:`echo 60*60*24*365*3 | bc` \
      VDEF:media2002=incidenti2002,AVERAGE \
      VDEF:max2002=incidenti2002,MAXIMUM \
      VDEF:mediana2002=incidenti2002,50,PERCENT \
      VDEF:media2003=incidenti2003,AVERAGE \
      VDEF:max2003=incidenti2003,MAXIMUM \
      VDEF:mediana2003=incidenti2003,50,PERCENT \
      VDEF:media2004=incidenti2004,AVERAGE \
      VDEF:max2004=incidenti2004,MAXIMUM \
      VDEF:mediana2004=incidenti2004,50,PERCENT \
      VDEF:media2005=incidenti2005,AVERAGE \
      VDEF:max2005=incidenti2005,MAXIMUM \
      VDEF:mediana2005=incidenti2005,50,PERCENT \
      LINE1:incidenti2002#0000ff:"Incidenti anno 2002" \
      LINE1:incidenti2003#00ff00:"Incidenti anno 2003" \
      LINE1:incidenti2004#ff0000:"Incidenti anno 2004" \
      LINE1:incidenti2005#121212:"Incidenti anno 2005" \
      COMMENT:"Confronto incidenti quadriennio 2002-2005" \
      GPRINT:media2002:"Media 2002 %4.0lf" \
      GPRINT:max2002:"Massimo 2002 %4.0lf" \
      GPRINT:mediana2002:"Mediana 2002 %4.0lf" \
      GPRINT:media2003:"Media 2003 %4.0lf" \
      GPRINT:max2003:"Massimo 2003 %4.0lf" \
      GPRINT:mediana2003:"Mediana 2003 %4.0lf" \
      GPRINT:media2004:"Media 2004 %4.0lf" \
      GPRINT:max2004:"Massimo 2004 %4.0lf" \
      GPRINT:mediana2004:"Mediana 2004 %4.0lf" \
      GPRINT:media2005:"Media 2005 %4.0lf     " \
      GPRINT:max2005:"Massimo 2005 %4.0lf     " \
      GPRINT:mediana2005:"Mediana 2005 %4.0lf"
```

Ed ecco infine il grafico risultante da questa analisi (lo script completo, dall'acquisizione alla creazione del grafico si trova in appendice):

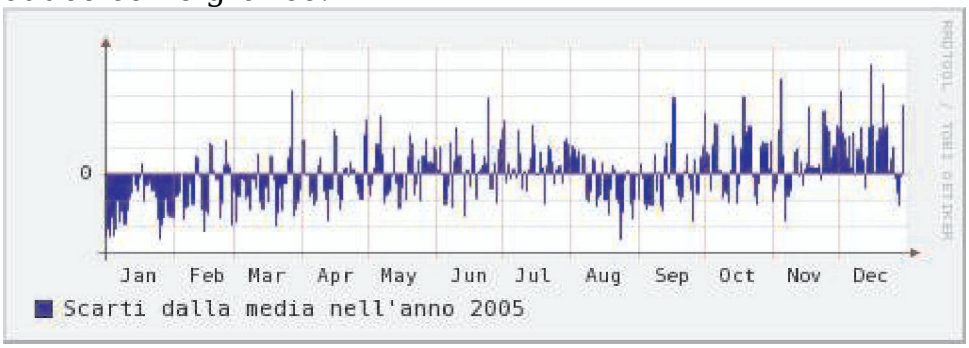


La visualizzazione dei dati in questa forma ci permette di evincere come ci sia stato effettivamente un abbassamento generalizzato del numero di incidenti dopo il secondo semestre del 2003 ma che la situazione ha ripreso a peggiorare nell'ultimo trimestre del 2005. E' evidente che una analisi completa non si possa limitare a questo grafico ma è altresì interessante constatare la forza espressiva di un grafico dettagliato che ci può indirizzare chiaramente verso le più opportune verifiche da effettuare sui dati; il fatto di poter rigenerare questo grafico in maniera pressoché istantanea rende chiaramente le potenzialità di analisi in “tempo reale” dello strumento.

Vediamo un ulteriore esempio che chiarifica l'uso delle variabili: rendiamo graficamente gli scostamenti delle media degli incidenti dell'ultimo anno per avere una prima visione dei momenti di maggiore pericolosità sulle strade:

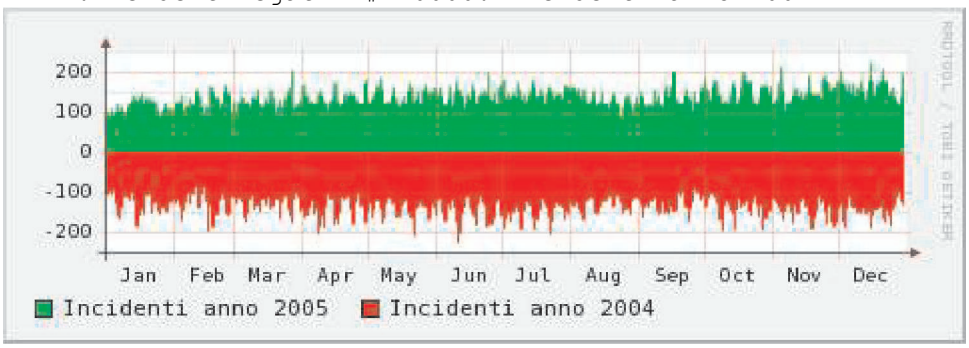
```
rrdtool graph polstrada_diff.png \
      --end `./date2ts 01/01/2006` \
      --start end-1y --step `echo 60*60*24 |bc` \
      DEF:incidenti=polstrada.rrd:incidenti_ss:AVERAGE: \
            end=`./date2ts 01/01/2006`:start=end-1y \
      VDEF:media=incidenti,AVERAGE \
      CDEF:scarti=incidenti,media,- \
      AREA:scarti#0000ff:"Scarti dalla media nell'anno 2005"
```

produce come grafico:



Confrontiamo infine il 2004 ed il 2005, creando un report dalla grafica più accattivante, invero non altrettanto di facile analisi:

```
rrdtool graph polstrada_45.png \
      --end `./date2ts 01/01/2006` \
      --start end-1y --step `echo 60*60*24*7|bc` \
      DEF:incidenti2004=polstrada.rrd:incidenti_ss:AVERAGE: \
            end=`./date2ts 01/01/2005`:start=end-1y \
      DEF:incidenti2005=polstrada.rrd:incidenti_ss:AVERAGE: \
            end=`./date2ts 01/01/2006`:start=end-1y \
      SHIFT:incidenti2004:`echo 60*60*24*365   | bc` \
      CDEF:incidentinegativi=-1,incidenti2004,*
      AREA:incidenti2005#00ff00:"Incidenti anno 2005" \
      AREA:incidentinegativi#ff0000:"Incidenti anno 2004"
```



6. Schedulazione delle attività

Abbiamo sin qui visto come acquisire, archiviare e rendere graficamente i dati da alcune pagine web: l'obiettivo di questa sezione è di indicare al lettore alcuni strumenti utili per automatizzare le operazioni di acquisizione dei dati, secondo le diverse necessità che nascono dal singolo caso in esame.

6.1 CRON

Cron è un demone[40] presente in tutti i sistemi *nix che permette di eseguire un comando periodicamente in intervalli di tempo predefiniti, stabilendo l'ora, i minuti, il giorno della settimana o il mese; l'operatività di cron dipende dal file crontab che definisce i momenti di attivazione ed i comandi da eseguire; ogni riga del crontab ha la seguente forma:

```
min hour dom month dow user command
```

dove i primi cinque valori indicano il momento dell'attivazione, il parametro “user” indica con quale utente deve essere eseguito il comando specificato nel parametro “command”; i primi cinque parametri indicano i minuti, l'ora, il giorno del mese (day of month), il mese ed il giorno della settimana per l'attivazione: se uno di questi parametri viene sostituito con un asterisco, il parametro non verrà considerato. Un altro caso particolare è specificare il parametro nella for-

40 I demoni sono tutti quei programmi che sono eseguiti senza nessuna interfaccia grafica in background; nei sistemi Windows sono tipicamente chiamati Servizi.

ma `*/num` che indica di eseguire il comando ogni “num” cambiamenti dello stato della variabile; l'ultimo caso particolare è l'utilizzo di una sequenza di numeri separati da virgole, che indica diversi momenti di attivazione; vediamo alcuni esempi per chiarire le idee e che potranno essere collegati facilmente ai nostri case study (il comando da eseguire è stato sostituito con la descrizione dell'operazione):

```
#esegue il comando al quarto d'ora di ogni ora
15 * * * * root “campanello una volta all'ora”
#esegue il comando ogni 5 minuti
*/5 * * * * minni “acquisisci dati della borsa”
#esegue il comando dal lunedì al venerdì a mezzanotte
0 0 * * 1,2,3,4,5 root “backup dei dati”
#esegue il comando ogni due giorni a mezzanotte
0 0 * * */2 root “aggiorna grafici polstrada”
```

La flessibilità e le potenzialità di questo comando sono elevatissime e ci permettono di effettuare uno scheduling molto preciso delle nostre attività; dobbiamo ricordarci che nel caso il nostro computer venisse spento, si interromperebbe anche il servizio di scheduling, con la conseguente assegnazione da parte di RRD di valori UNKNOWN nel momento dell'arresto: bisogna perciò focalizzare l'attenzione sulla reale possibilità della macchina che stiamo utilizzando di restare sempre accesa.

Tipicamente il comando cron viene gestito solamente dall'amministratore di sistema per la sua delicatezza e per le sue caratteristiche potenzialmente pericolose per il sistema (schedulando troppo spesso un lavoro molto oneroso si rischia, infatti, di compromettere, anche seriamente, le performance della macchina): questo fatto ne rende impossibile l'utilizzo in sistemi su cui non abbiamo un controllo completo o di cui non conosciamo l'amministratore.

6.2 AT

I comandi at ci permettono di eseguire una istruzione in un certo istante, specificando l'ora oppure il giorno:

```
at HHMM comando
at MMDDYY comando
```

Il comando sarà eseguito all'ora (HH) e minuti (MM) oppure nel giorno (DD), mese (MM) e anno (YY) definito con la stessa personalità dell'utente che ha lanciato il comando. Solitamente questo comando è ben tollerato dagli amministratori del sistema che lo lasciano a disposizione dell'utente, a differenza del precedente cron.

AT crea una coda di processi che può essere visualizzata con il comando `atq`: ad ogni elemento della coda viene associato un numero progressivo e, grazie a questo numero identificativo, l'operazione può essere rimossa con il comando `atrm`, cosiccome si opera con il set di comandi legati al demone di stampa lpd.

Per i nostri interessi potremmo pensare di invocare at al termine di ogni acquisizione per schedulare l'acquisizione successiva: un limite di questo approccio sta nel fatto che se il computer viene spento dovremmo preoccuparci di riavviare noi il comando oppure di inserire il comando stesso fra l'elenco dei comandi da eseguire all'avvio della macchina con le specifiche init.d, ad esempio, dei sistemi *nix; perciò risluterà più utile, per le nostre esigenze, utilizzare l'istruzione sleep, descritta in seguito, insieme allo scheduler di sistema[41].

41 Nei sistemi *nix e bsd lo scheduler è tipicamente il comando cron precedentemente citato, mentre nei sistemi Windows è chiamato “Pianificazione Attività”

6.3 Ricorsione ed iterazione

Se vogliamo far sì che i nostri script ripetano una serie di operazioni ad intervalli prefissati di tempo senza appoggiarci a nessuna applicazione esterna, possiamo utilizzare due strumenti che ci fornisce il PERL: la funzione `sleep` e la funzione `alarm`.

La funzione sleep può essere utilizzata all'interno di un ciclo e permette, come si intuisce dal nome, di attendere alcuni secondi prima di proseguire; ad esempio:

```
while () {
      sleep(15);
      print “Sono passati 15 secondi\n”;
}
```

La funzione alarm viene solitamente utilizzata per creare un timeout a delle operazioni di cui non possiamo conoscere la durata; un tipico esempio è:

```
my $timeout=15;
eval {
      local $SIG{ALRM} = \
            sub { die "Raggiunto TimeOut\n"; };
      alarm $timeout;
      #operazione onerosa e lunga
      alarm 0; #annulla l'allarme
};
```

Nel nostro caso potremmo inserire come funzione di attivazione il nostro script e come operazione da eseguire un ciclo infinito: questo approccio, seppur corretto dal punto di vista funzionale, risulta drasticamente più complesso del più semplice uso di sleep, nonché perverso poiché la funzione sleep è implementata usando appunto la stessa funzione alarm...

6.4 In conclusione: cosa è meglio usare?

Nel software che presenteremo nel prossimo capitolo e che vuole sintetizzare i concetti esposti sinora utilizzeremo solamente l'istruzione sleep in PERL, per motivi di semplicità di implementazione e di utilizzo. Gli utenti più esperti potranno fare a meno di utilizzare il parametro sleep e sono invece invitati ad utilizzare cron come strumento di scheduling: infatti implementare la ricorsività direttamente nello script prevede comunque di ricordarsi di attivare lo script ad ogni riavvio del sistema, cosiccome succederebbe utilizzando il comando at, mentre questi problemi vengono risolti utilizzando cron.

La precedente dichiarazione non può comunque essere utilizzata in maniera generale e dogmatica, ad esempio per l'acquisizione dei dati della borsa italiana si consiglia un misto di questi due strumenti: da una parte si consiglia di impostare il parametro sleep a cinque minuti (300 secondi), dall'altra di far si che l'iterazione si interrompa alle otto e mezza della sera (ovvero quando chiude la borsa); dall'altro lato inseriamo in crontab il comando per l'attivazione dello script quotidiano di acquisizione, dal lunedì al venerdì alle 8 di mattina:

```
#esegue il comando dal lunedì al venerdì a mezzanotte
0 8 * * 1,2,3,4,5 root “acquisici dati della borsa per 12 ore”
```

Il comando AT perde perciò rilevanza ma si è ritenuto presentarlo ugualmente per completezza di esposizione.

7 Mettiamo tutto insieme: WebAcquire

In questo capitolo analizzeremo e proporremo lo sviluppo di uno strumento che metta insieme tutte le nozioni finora apprese per recuperare determinate informazioni da specifiche pagine, per archiviarle e per renderle in maniera grafica.

Questo strumento si differenzia dai consueti crawler per la specificità del suo compito, che consiste essenzialmente nello scaricare un documento[42] dal web ed estrarne alcuni contenuti, eventualmente dopo aver seguito un percorso o aver effettuato qualche tipo di autenticazione e per archiviarne il risultato.

La specificità del suo compito, unita alla atomicità delle sue azioni, suggerisce di sviluppare questo elemento come una semplice applicazione a linea di comando altamente parametrizzabile anche attraverso file di configurazione; questa intuizione permette di sfruttare il componente in maniera molto semplice utilizzando i software di schedulazione visti in precedenza, nonché di sfruttare al meglio strutture di grid-computing nel caso di utilizzo intensivo, poiché l'applicazione principale non dovrà fare altro che inviare un processo ad ogni nodo e non sarà necessario intervenire sul codice per adattarlo alla struttura di calcolo su cui andrà ad operare.

42 Per documento di intende qualsiasi entità fruibile dal web, sia essa una pagina HTML, un feed RSS, un documento in PDF ecc.

7.1 Specifiche tecniche di webacquire

Webacquire accetterà in input un url di partenza, eventuali indicazioni su operazioni da eseguire dopo essersi collegati alla pagina (riempire campi di form, seguire link o analizzare frame particolari), una espressione regolare da applicare al documento ottenuto al termine del percorso, nonché alcune informazioni su come archiviare l'output del risultato ottenuto.

I parametri principali che il nostro componente dovrà recepire saranno::

- **configfile**, file di configurazione da cui leggere i parametri (deve essere possibile accettare anche più di un file di configurazione);
- **url**, url del documento iniziale a cui collegarsi;
- **action**, una o più azioni da eseguire dopo il collegamento: sono stringhe che nella forma "`azione|param1|param2|...`" che permettono di eseguire vari metodi di Mechanize;
- **regex**, espressione regolare da applicare alla documento acquisito a partire dall'url specificato e dopo aver compiuto le azioni necessaire;
- **variables**, espressione che permette di dare un nome agli elementi risultanti dall'applicazione dell'espressione regolare, ad esempio definire dato1 la variabile $1 e così via;
- **rrd_file**, nome del file RRD in cui archiviare il risultato della ricerca;
- **rrd_variables**, nome delle variabili da salvare nel file RRD, corrispondenti al nome dato alla sorgente dati nell'archivio

stesso;

- **file_name** e **file_content**, nome del file di testo e suo contenuto per archiviare in maniera classica i risultati.

Questi parametri regoleranno il funzionamento complessivo della applicazione e saranno affiancati da varie altre opzioni di carattere generale o più specifiche per i singoli compiti.

Bisogna prevedere la possibilità di effettuare un parsing dei parametri testuali più lunghi per sostituirne porzioni predeterminate con i valori relativi alla data di interesse o con dei valori passati a linea di comando; per fare ciò bisogna prevedere una funzione di sostituzione che sostituisca sequenze `[key]` con un determinato valore; key può rappresentare anche un componente della data: ad esempio [%Y] deve essere sostituito con il valore in quattro cifre dell'anno. I valori di sostituzione definiti nei file di configurazione devono, al contrario dei parametri, essere rimpiazzati dai valori a linea di comando in maniera che si possano creare dei file di configurazione generali da adattare con particolari parametri.

Deve essere possibile, inoltre, specificare una data iniziale di analisi ed anche una data finale, nonché un valore che ci dica come e quanto incrementare la data ad ogni passo successivo; bisogna poter specificare, inoltre, quanto tempo aspettare fra una acquisizione e la successiva, ovvero quali sistemi adottare per rendere meno invasivo il nostro processo di acquisizione.

Infine è necessario che l'applicazione esponga un help chiaro e facilmente leggibile, preferibilmente senza file di supporto.

7.2 Qualche altra libreria utile

Viste le specifiche tecniche definite, risulta necessario studiare come affrontare alcuni problemi: gestire i parametri di configurazione su file e a linea di comando, manipolare le date e, infine, formattare chiaramente l'help in linea.

7.2.1 Analisi dei Parametri

Per l'analisi dei parametri si utilizzerà la libreria AppConfig che permette di acquisire con le stesse modalità sia le opzioni passate a linea di comando, sia le opzioni presenti nei file di configurazione; per implementare l'acquisizione dei parametri si utilizza una struttura iterativa: non sfuggirà al lettore che sarebbe stato più consono utilizzare una struttura ricorsiva ma, d'altra parte, questa avrebbe complicato e reso di più difficile la lettura del codice, perciò si è preferito seguire la strada più semplice della iteratività; il primo passo è quello di definire i parametri disponibili, definendo inoltre se il parametro è sensibile alle lettere maiuscole (case-sensitive):

```
my $config = AppConfig->new({
        CASE    => 0,
        CREATE  => 1,
        },
        "help|!",
        "debug|d|verbose|v|!",
        "config|configfile|cfg|=s@",
        [...]
);
```

Per ogni parametro si elenca il nome e gli eventuali alias ed infine

si specifica la tipologia:

- **!** indica che il parametro non ha argomenti, la sua connotazione sarà vero o falso; ad esempio `-debug`
- **=s** indica che il parametro è valore unico, quindi passando due volte il parametro si otterrà come valore solo l'ultimo valore passato; ad esempio `-filename /etc/passwd`
- **=s@** indica che il parametro è una lista di valori, ovvero un array; ad esempio `-config conf1 -config conf2`
- **=s%** indica che il parametro è una lista di chiavi e valori; ad esempio `-date day=12 -date month=5 -date year=2005.`

Dopo aver impostato i valori di default per ogni parametro:

```
$config->debug(0);
$config->config('/etc/webacquire.conf');
$config->logfile('/tmp/webacquire.log');
[...]
```

carichiamo le opzioni passate via linea di comando:

```
$config->getopt();
```

e quindi effettuiamo il ciclo iterativo che ci permette di caricare tutte le opzioni dai file di configurazione eventualmente richiesti a linea di comando o richiesti nei file di configurazione stessi:

```
for (my $i=0;$i<scalar(@{$config->config()});$i++) {
        my @configs=@{$config->config()};
        my $configfile=$configs[$i];
        if (-e $configfile) {
                if (-r $configfile) $config->file($configfile);
        }
}
```

La struttura iterativa nasce dal fatto che ogni volta che acquisiamo un file di configurazione, `$config->file($configfile)`, se questo file contiene la definizione di altri file di configurazione, la dimensione dell'array dei parametri di configurazione aumenta e, di conseguenza, aumenta il numero di iterazioni totali compiute dal ciclo

principale, `scalar(@{$config->config()})`[43].

Sfruttando le potenzialità del parametro di tipo **s%**, ovvero lista di valori, definiamo un parametro **subst** di questo tipo per assolvere al compito di acquisire i dati per effettuare il parsing richiesto dalle specifiche; l'ultimo passaggio necessario per l'acquisizione dei parametri è il caricamento delle opzioni di sostituzione passate a riga di comando:

```
my %substs=%{$config->subst()};
for (my $i=0;$i<scalar(@ARGV);$i++) {
      my ($k, $v)=split(/=/, $ARGV[$i], 2);
      $substs{$k}=$v;
}
```

Nel seguito del codice si ignorerà la proprietà `subst` della classe `AppConfig` e si utilizzerà esclusivamente l'hash `substs` qui creato.

7.2.2 Utilizzo e manipolazione delle date

Per analizzare e manipolare le date si utilizzerà la libreria Date::Manip, che permette di analizzare una stringa di testo e tirarne fuori il suo valore come data; questa libreria ci consente di utilizzare anche la lingua italiana per effettuare l'analisi, nonché qualsiasi altra lingua o formato[44].

Per configurare questa libreria si utilizza il comando Date_Init, mentre per analizzare la data la funzione ParseDate; un'altra funzione importante è la funzione unixDate che permette di formattare in

43 Questa tecnica di acquisizione dei parametri è stata messa a punto ed utilizzata intensivamente già nel progetto MiNeT (Minimal Network Tool), un insieme di script per la creazione di file di configurazione e per l'analisi dei log.

44 Il formato di un data indica in che ordine sono specificati il giorno ed il mese: il formato americano prevede di specificare il mese seguito dal giorno, al contrario del formato europeo; perciò 3/2/2001 indicherà il due marzo in formato americano (US) ed il tre febbraio in formato europeo (non-US).

maniera opportuna una data; infine risulterà fondamentale per i nostri scopi la possibilità di sommare una data ad un intervallo di tempo, definito sempre in maniera discorsiva, per poter effettuare un ciclo di acquisizioni basato su un intervallo di date.

Vediamo un breve esempio del funzionamento:

```
use Date::manip;

Date_Init("Language=italian", "DateFormat=non-US");

my $today = ParseDate('oggi');
my $date_step = ParseDateDelta('- una settimana');
my $last_week = Datecalc($today, $date_step);

#stampa la data della setimana scorsa
#nel formato giorno/mese/anno
print unixDate($last_week, '%d/%m/%Y');
```

Risulta quindi indispensabile specificare dei nuovi parametri per la nostra applicazione:

date_language, lingua in cui saranno espresse le date, specificata in inglese, ad esempio italian per italiano (valore di default[45]);

date_format, formato con il quale saranno specificate le date, ovvero US (americano) o non-US (europeo, valore di default);

date_start, data iniziale in cui iniziare l'analisi (default: adesso);

date_stop, ultima data in cui specificare l'analisi (di default pari alla data iniziale);

date_delta, intervallo di tempo da aggiungere ad ogni passo alla data iniziale o del passo precedente (di default un mese);

45 Ovvero il valore assunto dal parametro quando questo non viene esplicitamente definito.

7.2.3 Formattazione dell'help in linea

Il linguaggio PERL permette di inserire grandi pezzi di documentazione all'interno del codice stesso: tutto ciò che è compreso fra una linea che inizia con =pod e la linea che inizia con =cut viene ignorato dal compilatore, ma può essere utilizzato per srivere documentazione tecnica.

Nei sistemi *nix esiste il comando man, che permette di visualizzare dei file di testo di documentazione in maniera graficamente uniforme, evidenziano la sintassi, la descrizione e l'utilizzo di un comando.

La libreria Pod::Usage mette insieme i due concetti fin qui espressi: permette di inserire all'interno di uno script PERL del testo formattato in maniera intellegibile dal comando man per visualizzarlo a richiesta; specificando l'opzione **help** nel nostro software si richiamerà, prima di terminare l'esecuzione, il comando pod2usage:

```
use Pod::Usage;
pod2usage(-exitstatus=>0, verbose=>2);

=head1 NAME
Nome dello script
=head1 SYNOPSYS
descrizione dello script
[...eccetera...]
=cut
```

7.3 Implementazione di WebAcquire

Ora abbiamo tutti gli elementi necessari per implementare il nostro software in tutte le sue parti ma, prima di analizzare la sua struttura, vediamo gli ulteriori parametri, non definite in precedenza, che lo caratterizzeranno:

- **debug**, fa sì che venga descritta puntualmente ogni azione intrapresa, in maniera da individuare eventuali errori nel codice o nella configurazione;
- **sleepy**, utilizza WWW::Mechanize::Sleepy in sostituzione di WWW::Mechanize;
- **sleep**, numero di secondi da attendere fra una acquisizione e la successiva (di default 1);
- **agent**, il “browser agent” da simulare, di default Mozilla Firefox in ambiente Linux/X11;
- **regex_cycle**, indica se effettuare la ricerca del risultato ciclicamente all'interno del documento risultante, invece di limitarsi alla sua prima occorrenza; se specificato non considera ulteriori espressioni regolari;
- **rrd_create_string**, specifica la linea di comando da utilizzare per creare l'archivio rrd; viene eseguita se l'archivio non è presente oppure se è specificato il parametro **rrd_recreate**;
- **rrd_graph_string**, specifica la linea di comando da utilizzare per creare il grafico rrd al termine dell'acquisizione;
- **rrd_update_time**, specifica, in formato discorsivo, la data con la quale saranno inseriti i dati nell'archivio RRD; se non è specificato nessun valore, viene utilizzata la data e l'ora corrente;

- **rrd_cumulate**, se specificato, esegue tutte le operazioni di aggiornamento dell'archivio dati RRD al termine del parsing completo dei dati; risulta molto utile quando si utilizza il parametro regex_cycle per evitare che RRD dia errore nel caso i dati nella pagina siano presentati in ordine cronologico discendente[46];
- **file_header**, l'intestazione da inserire nel file di testo alla sua creazione; il file viene creato se non è presente nel sistema ovvero se è specificato il parametro **file_recreate**;
- **number_italian2english**, indica di modificare i dati estratti dal documento, dal formato numerico italiano, con il punto come separatore delle migliaia e la virgola come separatore dei decimali, nel formato internazionale con il punto come separatore dei decimali.

Entriamo ora nel vivo del programma, vedendo innanzitutto come è stata implementata la funzione di parsing definita nelle specifiche:

```
#variabili globali di sostituzione
my %substs;
my $date_element="%y,%Y,%G,%L,%m,%f,%b,%B,%U,%W,%j,%d,%e,%v,%a,"
      "%A,%w,%E,%H,%k,%i,%I,%p,%M,%S,%s,%o,%Z,%z,%c,%C,%g,%D,"
      "%x,%l,%r,%R,%T,%V,%Q,%q,%P,%O,%F,%J,%K,%n,%t,%%,%+";
my $debug=0;

sub subst_string {
      my $text=shift;
      my $date=shift;

      print "Sostituzioni per\n\t$text\n" if ($debug);
      foreach my $key (keys(%substs)) {
            my $subst=$substs{$key};
            $text =~ s/\[$key\]/$subst/ig;
      }
      foreach my $frmt (split(/,/, $date_element)) {
            my $subst=UnixDate($date, $frmt);
            $text =~ s/\[$frmt\]/$subst/g;
      }
      print "\tRisultato:\n\t$text\n" if ($debug);
```

46 Ricordiamo che RRDTool non accetta che siano inseriti dati in ordine temporale casuale, bensì questi devono essere inseriti dal più vecchio al più giovane: se la pagina, ad esempio la Borsa Italiana, espone questi dati dal più nuovo al più vecchio, questa opzione risulta indispensabile.

```
      return $text;
}
```

La funzione accetta due parametri, il primo è il testo da analizzare mentre il secondo è la data da inserire eventualmente nelle chiavi di sostituzione inserite nel testo[47]; ogni parametro di sostituzione, specificato nei file di configurazione o a linea di comando, racchiuso fra parentesi quadrate, viene sostituito nel testo, cosiccome ogni elemento della data supportato dalla funzione UnixDate.

Il programma si snoda in maniera lineare: vengono inizialmente definite le librerie da utilizzare, i parametri globali, la funzione di sostituzione precedente, quindi vengono caricati i parametri di configurazione e gli opportuni valori di default; a questo punto bisogna verificare che siano stati specificati tutti i parametri minimi necessari per l'esecuzione, che sono l'url, l'espressione di ricerca, le variabili e la data iniziale:

```
#Variabili obbligatorie:
die("Specificare un URL\n") if ($config->url() =~ /^\s*$/ );
die("Specificare una espressione di ricerca\n")
      if ($config->regex() =~ /^\s*$/ );
die("Specificare le variabili\n")
      if ($config->variables() =~ /^\s*$/ );
die("Specificare la data iniziale\n")
      if ($config->date_start() =~ /^\s*$/ );
```

Se necessario vengono poi creati gli archivi testuali o RRD e quindi viene inizializzata la classe Mechanize, nella forma desiderata:

```
my $mech;
if ($config->sleepy()) {
```

47 Questa funzione risulta, in questa forma, decisamente con poche performance: per snellirla nella versione finale si utilizza il modificatore e nel secondo ciclo di ricerca, facendo così eseguire l'operazione UnixDate solamente se necessario: questa forma rende però praticamente illeggibile il codice e per questo motivo è stata omessa dal testo.

```
      $mech = WWW::Mechanize::Sleepy->new(
            agent=>$config->action());
} else {
      $mech = WWW::Mechanize->new(
            agent=>$config->action());
}
```

A questo punto, dopo aver controllato che le date fornite siano state specificate in una forma comprensibile da Date::Manip:

```
my $date_start = ParseDate($config->date_start()) or
      die('La data iniziale non e\' stata specificata correttamente');
```

non resta che iniziare il nostro lavoro. Innanzitutto si inizierà un ciclo che si interromperà non appena la data corrente (inizialmente pari al parametro date_start) non avrà superato la data conclusiva:

```
while (Date_Cmp($date_curr, $date_stop)<=0) {
      [...]
}
```

Ad ogni iterazione ci si connette all'url specificato, dopo aver eseguito l'operazione di sostituzione delle chiavi conn subst_string, e si eseguiranno le eventuali azioni azioni specificate nella forma `"azione|parametro1|parametro2|..."`, ad esempio `"submit_form|username|tomaso|password|segreta"`:

```
my $url=subst_string($config->url(), $date_curr);
$mech->get($url);

#Eseguo le azioni
my @actions=@{$config->action()};
for (my $i=0; $i<scalar(@actions); $i++) {
      my $action=subst_string($actions[$i], $date_curr);

      my @elements = split(/\|/, $action);
      if ($elements[0] =~ /submit_form/) {
            my $form_number=$elements[1];
            $mech->form_number($form_number);
            for (my $e=2;$e<scalar(@elements);$e+=2) {
                  my $k=$elements[$e];
                  my $v=$elements[$e+1];
                  $mech->field($k, $v);
            }
            $mech->submit();
      } elsif ($elements[0] =~ /follow_url$/) {
            my $link=$elements[1];
            $mech->follow_link( url => $link );
      } elsif ($elements[0] =~ /follow_url_regex$/) {
```

```
            my $link=$elements[1];
            $mech->follow_link( url_regex => qr/$link/i );
      } else {
            print "\nAzione NON riconosciuta ".$elements[0]."\n";
      }
}

my $html = $mech->content();
```

A questo punto il codice HTML viene analizzato con le espressioni regolari (con un ciclo o una espressione regolare alla volta a seconda se è stato specificato il parametro regex_cycle) e quindi viene valorizzato l'hash ref che sarà utilizzato per salvare i file nell'archivio RRD o nel file di testo:

```
[..]
my %dati;
my $founded=0;
for (my $i=0;$i<scalar(@regexs);$i++) {
      my $regex=subst_string($regexs[$i], $date_curr);
      #il modificatore s permettedi considerare il documento
      #come su una unica riga (per far si che .* comprenda anche gli
      #a-capo, cosiccome \s
      my @variables=split(/,/, $vars[$i]);
      if ($html =~ /$regex/s) {
            my $j=1;
            foreach my $tmp (@variables) {
                  my $tmp2=eval('$'.$j);
                  $dati{$tmp}=$tmp2;
                  $j++;
            }
            $founded=1;
      } else {
            print "Espressione $regex non trovata\n" if ($debug);
      }
}

if ($founded) {
      [...]
      if ($file_save) {
            open(OUTPUT, ">>$file_name");
            my $line=subst_string($file_content, $date_curr);
            my $line=eval($line);
            print OUTPUT "$line\n";
            close(OUTPUT);
      }
      [...]
      unless ($rrd_variables =~ /^\s*$/) {
            foreach my $tmp (keys(%dati)) {
                  unless ($rrd_variables =~ /,$tmp,/) {
                        delete $dati{$tmp};
                  }
```

```
            }
      }
      $rrd->update(time => UnixDate($date_curr, "%s"),
            values => \%dati) if ($rrd_save);
```

Si noti l'utilizzo della funzione eval: l'istruzione

```
my $line=eval($line);
```

permette di far sì che la stringa line sia trattata come una espressione PERL, cosicché definendola, ad esempio:

```
line='"$dati{campo1}\t$dati{campo2}"'
```

dopo questa operazione contenrrà esattamente i valori degli elementi campo1 e campo2 dell'hash dati.

A questo punto non resta altro che incrementare la data corrente dell'intervallo specificato ed aspettare alcuni secondi prima di ripetere il ciclo di acquisizione, manipolazione ed archiviazione dati:

```
my $err;
$date_curr = DateCalc($date_curr,$date_delta,\$err);
die "\n\nErrore ($err) nell'incremento della ".
      "data [".$config->date_delta()."]\n\n" if ($err);

sleep($step_wait) if ($step_wait);
```

Il controllo di errore inserito è sovrabbondante rispetto al controllo iniziale, ma si è ritenuto di inserirlo nuovamente per puro scrupolo.

Al termine di tutti i cicli, ovvero quando la data corrente supera la data finale, si esegue, se specificata, l'operazione di creazione del grafico, ovvero qualsiasi altro comando di sistema:

```
my $rrd_graph=$config->rrd_graph_string();
unless ($rrd_graph=~/^\s*$/) {
      my $result=`$rrd_graph`;
}
```

7.4 Configurare WebAcquire

Vediamo ora come configurare WebAcquire per portare a termine i compiti che ci siamo posti come obbiettivo nei case study: trascureremo nel testo i parametri per la creazione dell'archivio RRD e dei grafici, già discussi in precedeza (le configurazioni complete sono riportate in appendice).

I dati della polstrada sono inseriti ed accessibili dal primo marzo 2003; vediamo come acquisire tutti i dati sino ad oggi:

```
rrd_recreate=on
rrd_file=polstrada.rrd

date_start='01/03/2001'
date_stop='ieri'
date_delta='+1g'

url = "http://www.poliziadistato.it/pds/stradale/archivio/zoom.php?vg=
[%d].[%m].[%Y]"

subst reg_cella_piena='<td.*?>.*?<span.*?>(\d*)<\/span>.*?<\/td>.*?'
subst reg_cella_vuota='<td.*?>.*?<\/td>.*?'

regex='Totale delle pattuglie impiegate.*?<\/td>.*?[reg_cella_piena]
[reg_cella_piena]'
variables='pattuglie_as,pattuglie_ss'

[...]
```

In questo esempio vediamo come possiamo utilizzare i parametri subst anche per rendere più leggibile lo stesso file di configurazione (dal quale sono state omesse, per brevità, tutte le altre espressioni di ricerca). Vediamo, inoltre, come sia possibile definire nella forma discorsiva l'incremento della data in maniera sintetica: +1g significa “aggiungi un giorno”. Per eseguire lo script sarà sufficiente scrivere:

```
webacquire -conf polstrada.con
```

I dati della Borsa Italiana vanno acquisiti ogni cinque minuti,

sfruttando tutto il contenuto della pagina; vanno inoltre archiviati solo al termine dell'acquisizione poiché vengono dati in ordine decrescente:

```
rrd_file=borsa.rrd
rrd_variables='valore'
rrd_update_time='[%d]/[%m]/[%Y],$dati{ore}:$dati{minuti}:00'
rrd_cumulate=on

date_start='adesso'
date_stop='oggi,20:30'
date_delta='+5minuti'

sleep=300
number_italian2english=on

url="http://www.borsaitalia.it/bitApp/intraday.bit?target=indici&isin=
II935&lang=it"

regex_cycle=on
regex='<\!\-\- IF class=tdazz\/tdbia \-\->\s*<tr class\="(tdazz|
tdbia)">\s*<td class\="dato">(\d{2})\.(\d{2})\.00<\/td>\s*<td
class\="dato">([0-9.,]*)</td>\s*<\!\-\- IF class=red\/green\/blu \-\->
\s*<td class\="(red|green|blu)">([0-9.,\-\+]*)<\/td>\s*<td
class\="dato">([0-9]*)<\/td>\s*</tr>'
variables='temp1,ore,minuti,valore'
```

in questo caso si sarebbe rivelato inutile specificare più di una espressione regolare poiché, avendo attivato l'opzione regex_cycle, qualsiasi altro parametro sarebbe stato ignorato; per attivare una opzione si può specificarla così come è, oppure impostarla al valore on (come nell'esempio), al valore true o a yes.

Vediamo, infine, come recuperare ed archiviare i dati forniti dall'ARPA emiliana all'ultimo giono disponibile:

```
file_name='arpa.csv'
file_recreate=off
file_header='"Anno\tMese\tGiorno\tCarpi\tFiorano\tModena1\tModena2\tMo
dena3"'
file_content='"$dati{anno}\t$dati{mese}\t$dati{giorno}\t$dati{fiorano}
\t$dati{carpi}\t$dati{modena1}\t$dati{modena2}\t$dati{modena3}"'

date_start='adesso'

url = "http://service.arpa.emr.it/qualita-aria-
```

```
2005/bollettino.aspx?prov=mo"

#recupera il giorno
regex='<option selected="selected" value="(\d\d)\/(\d\d)\/(\d\d\d\d)"'
variables='giorno,mese,anno'

#recupera i valori
regex='CARPI \- CARPI 2 \(VIA REMESINA \)<\/td><td>(.*?)<\/td>'
variables='carpi'
[...]
```

questa configurazione permette di inserire nel file di testo i dati dell'ultimo giorno disponibile, sfruttando una espressione regolare che analizza l'elemento attico della combobox di selezione del giorno. In questa maniera scavalchiamo i problemi di analisi che avevamo posto nel testo ma che ci sono serviti esclusivamente per motivi didattici. Per far si che questa operazione venga eseguita ogni giorno alla mezzanotte e che ci si crei quindi un corretto archivio dati, inseriamo nel file crontab la seguente riga:

```
0 0 * * ** root "webacquire -conf /etc/webacquire/arpa.conf"
```

Appendice 1: Heartbeat e RRD

Ogni archivio RRD è caratterizzato dal parametro “step”, ovvero ogni quanti secondi viene fornito un dato; alla sua creazione l'archivio risulta impostato in ogni punto con il valore “UNKNOWN” che verrà man mano sostituito dai valori forniti ai “data sources”.

Per ogni “data source” viene definito il parametro “heartbeat” che rappresenta il massimo intervallo accettabile fra due valori noti: se la distanza fra due valori noti è minore di questo parametro, allora la media fra questi valori viene inserita nell'intervallo, altrimenti l'intero intervallo viene riempito dal valore “UNKNOWN”; si capisce quindi l'importanza del valore di questo parametro che non definisce semplicemente il massimo intervallo accettabile fra due valori, bensì definisce il massimo numero di “UNKNOWN” consecutivi nel campione, la quantità oltre la quale non è possibile effettuare nessuna interpolazione.

Per i nostri scopi il valore dell'”heartbeat” sarà abbastanza ininfluente, poiché recuperando i dati dal web in precisi intervalli non dovrebbe essere possibile creare dei “buchi informativi”, anzi mantenere un basso “heartbeat”, magari il doppio del valore dello “step”, significa avere un ulteriore sistema di controllo per verificare la funzionalità del nostro software: poiché, ad esempio, i dati della Borsa Italiana ci interessa ottenerli ogni cinque minuti per esporre grafici dettagliati, la mancanza di pochi campioni comprometterebbe il valore del nostro lavoro, riportando i grafici all'approssimazione già fornita.

Appendice 2: Mechanize in dettaglio

Mechanize nasce dalla fusione delle librerie LWP ed HTML::Parser: LWP[48] è un insieme di procedure per accedere ai servizi del Web, ovvero una serie di funzioni per implementare completi software client WWW e semplici applicazioni server HTTP; HTML::Parser, già citata in queste pagine, permette di effettuare una analisi molto dettagliata della struttura di una pagina web.

Mechanize ci permette di navigare nelle pagine web come se stessimo utilizzando un vero e proprio browser: non dovremmo, infatti, preoccuparci della gestione dei cookies, dei campi nascosti oppure di specificare da quale url proveniamo; il browser sarà però parzialmente limitato, non riuscendo a gestire pagine sviluppate in flash o con altri plugin. I browser moderni permettono di inserire all'interno delle pagine del codice JavaScript client-side: cosa succede in questo caso con Mechanize? Normalmente Mechanize non riconosce il codice Javascript e lo ignora, ma se risultasse necessario possiamo utilizzare due suoi illustri figli: Win32::IE::Mechanize e Mozilla::Mechanize, che ci permettono di utilizzare le librerie di parsing native dei due più diffusi browser, eliminando quindi qualsiasi differenza fra la navigazione “virtuale” e quella “reale”.

Tipicamente il nostro scopo sarà di riempire un form di una pagina web oppure seguire alcuni link in essa presenti ed analizzarne il risultato. Abbiamo visto nel testo alcuni script di esempio, vediamo ora di descrivere alcuni dei metodi più utili di questa libreria.

48 LWP è l'acronimo per libwww-perl, ovvero libreria Perl per il WWW

I metodi `agent_alias` e `known_agent_aliases` ci permettono di definire quale browser dovremmo simulare, dandoci anche un elenco dei browser conosciuti; ad esempio invece di inizializzare la classe con

```
my $mech = WWW::Mechanize->new(
      agent=>“Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)”
);
```

sarà sufficiente definire

```
$mech->agent(“Windows IE6”);
```

Per selezionare un form presente in una pagina si possono usare i metodi `form_number` e `form_name`: il primo risulta particolarmente utile quando nel form che ci interessa non è stato definito l'attributo nome; per impostare i valori dei vari campi del form si può utilizzare la generica funzione `field` (o l'equivalente `set_fields` per utilizzare variabili hashref), oppure addirittura riempire semplicemente i campi visibili, ignorando nome, form e posizione, esattamente come se utilizzassimo semplicemente il tasto tabulatore in un browser grafico, con il comando `set_visible`. Nel caso ci interessasse soffermarci di più sulla struttura dei campi del form, potremo usare le funzioni di riempimento specifiche per ogni tipo:

- **select** permette di scegliere selezionare un valore da una lista a discesa o di attivare uno o più elementi di una lista,
- **tick** permettere di vistare un checkbox
- **untick** permette di togliere la vista ad un checkbox
- **click** permette di premere un pulsante e, nel caso sia grafico, di specificare opzionalmente le coordinate del cursore (si pensi alla selezione di un'area da una mappa)
- **click_button** simula la pressione di un bottone, specificandone il nome, il numero oppure il valore

- **submit** preme il pulsante submit

Per seguire un link all'interno della pagina si può utilizzare il metodo `follow_link` che permette di scegliere il collegamento in base:

- al suo url di destinazione (url)
- all'url assoluto di destinazione (url_abs)
- al testo del collegamento (text)
- al nome del collegamento (name)
- al tipo di tag in cui è specificato il collegamento (tag), ad esempio un link (a) o un frame

Ogni elemento può essere individuato anche attraverso il suo matching con una espressione regolare, aggiungendo la particella regex in appendice al nome dell'opzione, ad esempio url_abs_regex.

Per creare script perl con mechanize si può utilizzare la classe HTTP::Recorder che permette di creare un proxy che intercetta la nostra navigazione e che produce il codice Mechanize corretto per la sua riproduzione in seguito. Il codice per creare il server proxy risulta molto semplice:

```
#!/usr/bin/perl
use HTTP::Proxy;
use HTTP::Recorder;

my $proxy = HTTP::Proxy->new();

# create a new HTTP::Recorder object
my $agent = new HTTP::Recorder;

# set the log file (optional)
$agent->file("/tmp/myfile");
```

```
# set HTTP::Recorder as the agent for the proxy
$proxy->agent( $agent );

# start the proxy
$proxy->start();

1;
```

Il funzionamento di HTTP::Recorder è quanto mai stupefacente ma denota seri problemi navigando su servizi web particolarmente elaborati, come le applicazioni Oracle Web Forms che modificano, ad esempio, la variabile di sessione ad ogni accesso.

Un altro strumento importante per imparare ad utilizzare Mechanize è la libreria WWW::Mechanize::Shell che ci permette di eseguire comandi di Mechanize sintetici in un ambiente a linea di comando; la shell viene attivata scrivendo.

```
perl -MWWW::Mechanize::Shell -eshell
```

ed a questo punto i comandi da utilizzare sono esattamente i nomi dei metodi visti in precedenza, ovvero allorché scrivevamo:

```
$mech->url("http://qualcosa.it");
```

ora scriveremo

```
url http://qualcosa.it
```

Per concludere questa presentazione è importante ricordare che l'acquisizione di dati da un servizio web con strumenti automatici non è equivalente alla navigazione, sia da un punto di vista legale che di netiquette: si consiglia quindi di fare sempre riferimento ai termini ed alle condizioni di utilizzo esposte dal webmaster in un link normalmente presente in tutte le pagine e di analizzare il contenuto dei robots[49] inseriti. Infine, nel caso di analisi molto pesanti e di siti

49 Con robots si intende il file robots.txt inserito nella radice del sito; le specifiche dei robots certificati dal W3Consortium si possono reperire su http://www.robotstxt.org/wc/robots.html

non troppo “robusti” dal punto di vista dell'hardware e della connessione, nonché per rendere meno invasiva, fastidiosa ed individuabile la nostra analisi per gli amministratori del sistema, si consiglia di utilizzare di preferenza WWW::Mechanize::Sleepy che fa una valutazione del tempo di battitura dei comandi equivalenti in un browser per rallentare le richieste successive così da rendere più credibile, trasparente e leggero il nostro passaggio.

Appendice 3: Script introduttivi

Esempi del''utilizzo di CURL e Mechanize

Polizia Stradale

```
use WWW::Curl::Easy;
my $GG='05';
my $MM='05';
my $AAAA='2006';
my $page_url="http://www.poliziadistato.it/pds/stradale/archivio”.
      “/zoom.php?vg=${GG}.${MM}.${AAAA}";

my $curl = new WWW::Curl::Easy;
$curl->setopt(CURLOPT_URL, $page_url);
$curl->perform;
print $curl->getinfo(CURLINFO_HTTP_CODE);
```

Borsa Italiana

```
use WWW::Curl::Easy;
my $page_url="http://www.borsaitalia.it/bitApp/intraday.bit?”.
      “isin=II999&target=indici&lang=it&from=1030&”.
      “querymode=byTime&HH=10&MM=30";

my $curl = new WWW::Curl::Easy;
$curl->setopt(CURLOPT_URL, $page_url);
$curl->perform;
print $curl->getinfo(CURLINFO_HTTP_CODE);
```

ARPA Emilia Romagna con CURL

```
use WWW::Curl::Easy;
my $GG='05';
my $MM='05';
my $AAAA='2006';
my $page_url="http://service.arpa.emr.it/qualita-aria-2005/”.
      “bollettino.aspx?prov=mo";

my $curl = new WWW::Curl::Easy;
$curl->setopt(CURLOPT_URL, $page_url);
$curl->setopt(CURLOPT_POSTFIELDS, "__EVENTTARGET=drpData");
$curl->setopt(CURLOPT_POSTFIELDS, "__EVENTARGUMENT=");
$curl->setopt(CURLOPT_POSTFIELDS, "__VIEWSTATE=");
$curl->setopt(CURLOPT_POSTFIELDS, "drpData=${GG}/${MM}/${AAAA}");
```

```
$curl->perform;
print $curl->getinfo(CURLINFO_HTTP_CODE);
```

ARPA Emilia Romagna con Mechanize

```
#!/usr/bin/perl

use WWW::Mechanize;

$agent="Mozilla/5.0 (X11; U; Linux i686; it-IT; rv:1.7.12) ".
      "Gecko/20051010 Firefox/1.0.7";
$url="http://service.arpa.emr.it/qualita-aria-2005/".
      "bollettino.aspx?prov=mo";

my $mech = WWW::Mechanize->new(
        agent=>$agent
);

#my $curl = new WWW::Curl::Easy;
#$curl->setopt(CURLOPT_URL, $page_url);
#$curl->setopt(CURLOPT_POSTFIELDS, "__EVENTTARGET=drpData");
#$curl->setopt(CURLOPT_POSTFIELDS, "__EVENTARGUMENT=");
#$curl->setopt(CURLOPT_POSTFIELDS, "__VIEWSTATE=");
#$curl->setopt(CURLOPT_POSTFIELDS, "drpData=${GG}/${MM}/${AAAA}"

$mech->get($url);
$mech->form_number($form_number);
$mech->select('drpData', '05/05/2006');
#$mech->field('drpData', '20/04/2006');
$mech->submit();

print $mech->content();
```

Esempi sulle Espressioni Regolari

Polizia Stradale

```
use strict;
use WWW::Mechanize;
use Data::Dumper;

my $agent="Mozilla/5.0 (X11; U; Linux i686; it-IT; rv:1.7.12) ".
      "Gecko/20051010 Firefox/1.0.7";
my $GG='05';
my $MM='05';
my $AAAA='2006';
my $url = "http://www.poliziadistato.it/pds/stradale/".
      "archivio/zoom.php?vg=${GG}.${MM}.${AAAA}";

my $mech = WWW::Mechanize->new(
```

```
        agent=>$agent
);

$mech->get($url);

my $html=$mech->content();

my $reg_cella_piena='<td.*?>.*?<span.*?>(\d*)<\/span>.*?<\/td>.*?';
my $reg_cella_vuota='<td.*?>.*?<\/td>.*?';

my $reg='Totale delle pattuglie impiegate.*?<\/td>.*?'.
       $reg_cella_piena.
       $reg_cella_piena;

$html =~ /$reg/s;
print "Pattuglie: autostrade $1, altre $2\n";

my $reg='Totale incidenti\s+rilevati.*?<\/td>.*?'.
       $reg_cella_piena.
       $reg_cella_piena;

$html =~ /$reg/s;
print "Incidenti: autostrade $1, altre $2\n";
```

Borsa Italiana, singola estrazione

```
use WWW::Mechanize;
use Data::Dumper;

$agent="Mozilla/5.0 (X11; U; Linux i686; it-IT; rv:1.7.12) ".
      "Gecko/20051010 Firefox/1.0.7";
$url='http://www.borsaitalia.it/bitApp/intraday.bit?'.
      'isin=II999&target=indici&lang=it&from=1030&'.
      'querymode=byTime&HH=10&MM=30';

my $mech = WWW::Mechanize->new(
        agent=>$agent
);

$mech->get($url);

$html=$mech->content();

$reg='<td>Ora<\/td>\s*'.
       '<td>Ultimo Valore<\/td>\s*'.
       '<td>Var \%<\/td>\s*'.
       '<td>Progressivo<\/td>\s*'.
       '<\/tr>.*?<tr class="tdazz">\s*'.
       '<td class="dato">(\d{2})\.(\d{2}).*?<\/td>\s*'.
       '<td class="dato">([0-9.,]*)<\/td>';

print "\nTrovata!\n" if ($html =~ /$reg/s);
$ore=$1;
```

```
$minuti=$2;
$datog=$3;
$dato=$3;
$dato =~ s/\.//;
$dato =~ s/\,/./;

print "\n\nOre $ore, Minuti $minuti, Dato Grezzo $datog, Dato
$dato\n\n";
```

Borsa Italiana, estrazione multipla

```
use WWW::Mechanize;
use Data::Dumper;
use strict;

my $agent="Mozilla/5.0 (X11; U; Linux i686; it-IT; rv:1.7.12) ".
      "Gecko/20051010 Firefox/1.0.7";
my $url='http://www.borsaitalia.it/bitApp/intraday.bit?'.
      'isin=II999&target=indici&lang=it&from=1030&querymode=byTime&HH=
10&MM=30';

my $mech = WWW::Mechanize->new(
        agent=>$agent
);

$mech->get($url);

my $html=$mech->content();

#$reg='<td>Ora<\/td>\s*'.
#      '<td>Ultimo Valore<\/td>\s*'.
#      '<td>Var \%<\/td>\s*'.
#      '<td>Progressivo<\/td>\s*'.
#      '<\/tr>.*?<tr class="tdazz">\s*'.
#      '<td class="dato">(\d{2})\.(\d{2}).*?<\/td>\s*'.
#      '<td class="dato">([0-9.,]*)<\/td>';
my $reg='<\!\-\- IF class=tdazz\/tdbia \-\->\s*'.
       '<tr class\="(tdazz|tdbia)">\s*'.
          '<td class\="dato">(\d{2})\.(\d{2})\.00<\/td>\s*'.
          '<td class\="dato">([0-9.,]*)</td>\s*'.
          '<\!\-\- IF class=red\/green\/blu \-\->\s*'.
          '<td class\="(red|green|blu)">([0-9.,\-\+]*)<\/td>\s*'.
          '<td class\="dato">([0-9]*)<\/td>\s*'.
       '</tr>';

my @result;
my $i=0;
while ($html =~ /$reg/gs) {
      $result[$i]={
            'ore'=>$2,  'minuti'=>$3,
            'dato'=>$4, 'perc'=>$6, 'prog'=>$7
      };
```

```
      print "$result[$i]{ore}:$result[$i]{minuti}: ".
            "Dato $result[$i]{dato} - Prec $result[$i]{perc} - ".
            "Prog $result[$i]{prog}\n";
      $i++;
}
```

ARPA Emilia Romagna

```
use WWW::Mechanize;
use Data::Dumper;

$agent="Mozilla/5.0 (X11; U; Linux i686; it-IT; rv:1.7.12) ".
      "Gecko/20051010 Firefox/1.0.7";
$url="http://service.arpa.emr.it/qualita-aria-
2005/bollettino.aspx?prov=mo";

my $mech = WWW::Mechanize->new(
        agent=>$agent
);

#my $curl = new WWW::Curl::Easy;
#$curl->setopt(CURLOPT_URL, $page_url);
#$curl->setopt(CURLOPT_POSTFIELDS, "__EVENTTARGET=drpData");
#$curl->setopt(CURLOPT_POSTFIELDS, "__EVENTARGUMENT=");
#$curl->setopt(CURLOPT_POSTFIELDS, "__VIEWSTATE=");
#$curl->setopt(CURLOPT_POSTFIELDS, "drpData=${GG}/${MM}/${AAAA}"

$mech->get($url);
$mech->form_number($form_number);
$mech->select('drpData', '05/05/2006');
#$mech->field('drpData', '20/04/2006');
$mech->submit();

$html=$mech->content();

$html =~ /MODENA \- MO \- PARCO FERRARI \( PARCO FERRARI \)<\/td><td>
(.*?)<\/td>/i;
$pm_10_ferrari = $1;
$html =~ /MODENA \- MO \- VIA GIARDINI \(VIA GIARDINI \)<\/td><td>
(.*?)<\/td>/i;
$pm_10_giardini = $1;
$html =~ /MODENA \- MO \- VIA NONANTOLANA \(VIA CIMONE \)<\/td><td>
(.*?)<\/td>/i;
$pm_10_nonantolana = $1;

print "\n\nParco Ferrari:   $pm_10_ferrari\n".
            "Via Giardini:    $pm_10_giardini\n".
            "Via Nonantolana: $pm_10_nonantolana\n";
```

Archiviazione Dati su RRD: Polizia di Stato

```
use strict;
use WWW::Mechanize;
use Data::Dumper;
use Date::Manip;

use RRDTool::OO;

#Create an interface object
my $rrd = RRDTool::OO->new(
      file => "polstrada.rrd" );

my $agent="Mozilla/5.0 (X11; U; Linux i686; it-IT; rv:1.7.12) ".
      "Gecko/20051010 Firefox/1.0.7";
my $GG='05';
my $MM='05';
my $AAAA='2006';
my $url = "http://www.poliziadistato.it/pds/stradale/".
      "archivio/zoom.php?vg=${GG}.${MM}.${AAAA}";

my $mech = WWW::Mechanize->new(
        agent=>$agent
);

print "Acquisisco i dati\n";
$mech->get($url);

my $html=$mech->content();

print "Dati Acquisiti\n";
my %dati;

my $reg_cella_piena='<td.*?>.*?<span.*?>(\d*)<\/span>.*?<\/td>.*?';
my $reg_cella_vuota='<td.*?>.*?<\/td>.*?';

my $reg='Totale delle pattuglie impiegate.*?<\/td>.*?'.
       $reg_cella_piena.
       $reg_cella_piena;
$html =~ /$reg/s;
$dati{'pattuglie_as'}=$1;
$dati{'pattuglie_ss'}=$2;
print "Pattuglie: autostrade $1, altre $2\n";

my $reg='Totale incidenti\s+rilevati.*?<\/td>.*?'.
       $reg_cella_piena.
       $reg_cella_piena;
$html =~ /$reg/s;
$dati{'incidenti_as'}=$1;
$dati{'incidenti_ss'}=$2;
print "Incidenti: autostrade $1, altre $2\n";

&Date_Init("Language=Italian","DateFormat=non-US");
my $date = ParseDate("$GG/$MM/$AAAA");
```

```
$rrd->update(time => UnixDate($date, "%s"),
      values => \%dati);
```

Dall'acquisizione al grafico: esempio completo

```
use strict;
use WWW::Mechanize;
use Data::Dumper;
use Date::Manip;

use RRDTool::OO;

my $rrd_create="rrdtool create polstrada.rrd ".
      "--step `echo 60*60*24 | bc` ".
      "--start `./date2ts 28/02/2001` ".
      "DS:pattuglie_as:GAUGE:172800:0:5000 ".
      "DS:incidenti_as:GAUGE:172800:0:1000 ".
      "DS:imortali_as:GAUGE:172800:0:1000 ".
      "DS:iferiti_as:GAUGE:172800:0:1000 ".
      "DS:deceduti_as:GAUGE:172800:0:1000 ".
      "DS:feriti_as:GAUGE:172800:0:1000 ".
      "DS:pattuglie_ss:GAUGE:172800:0:5000 ".
      "DS:incidenti_ss:GAUGE:172800:0:1000 ".
      "DS:imortali_ss:GAUGE:172800:0:1000 ".
      "DS:iferiti_ss:GAUGE:172800:0:1000 ".
      "DS:deceduti_ss:GAUGE:172800:0:1000 ".
      "DS:feriti_ss:GAUGE:172800:0:1000 ".
      "RRA:AVERAGE:0.5:1:3650 ".
      "RRA:AVERAGE:0.5:1:28 ".
      "RRA:AVERAGE:0.5:1:90 ".
      "RRA:AVERAGE:0.5:7:52 ".
      "RRA:AVERAGE:0.5:7:150";

my $rrd_graph='rrdtool graph polstrada.png '.
      '--end `./date2ts 01/01/2006` --start end-1y --step `echo
60*60*24*7|bc` '.
      'DEF:incidenti2002=polstrada.rrd:incidenti_ss:AVERAGE:'.
            'end=`./date2ts 01/01/2003`:start=end-1y '.
      'DEF:incidenti2003=polstrada.rrd:incidenti_ss:AVERAGE:'.
            'end=`./date2ts 01/01/2004`:start=end-1y '.
      'DEF:incidenti2004=polstrada.rrd:incidenti_ss:AVERAGE:'.
            'end=`./date2ts 01/01/2005`:start=end-1y '.
      'DEF:incidenti2005=polstrada.rrd:incidenti_ss:AVERAGE:'.
            'end=`./date2ts 01/01/2006`:start=end-1y '.
      'SHIFT:incidenti2004:`echo 60*60*24*365   | bc` '.
      'SHIFT:incidenti2003:`echo 60*60*24*365*2 | bc` '.
      'SHIFT:incidenti2002:`echo 60*60*24*365*3 | bc` '.
      'VDEF:media2002=incidenti2002,AVERAGE '.
      'VDEF:max2002=incidenti2002,MAXIMUM '.
      'VDEF:mediana2002=incidenti2002,50,PERCENT '.
      'VDEF:media2003=incidenti2003,AVERAGE '.
      'VDEF:max2003=incidenti2003,MAXIMUM '.
      'VDEF:mediana2003=incidenti2003,50,PERCENT '.
```

```
      'VDEF:media2004=incidenti2004,AVERAGE '.
      'VDEF:max2004=incidenti2004,MAXIMUM '.
      'VDEF:mediana2004=incidenti2004,50,PERCENT '.
      'VDEF:media2005=incidenti2005,AVERAGE '.
      'VDEF:max2005=incidenti2005,MAXIMUM '.
      'VDEF:mediana2005=incidenti2005,50,PERCENT '.
      'LINE1:incidenti2002#0000ff:"Incidenti anno 2002" '.
      'LINE1:incidenti2003#00ff00:"Incidenti anno 2003" '.
      'LINE1:incidenti2004#ff0000:"Incidenti anno 2004" '.
      'LINE1:incidenti2005#121212:"Incidenti anno 2005" '.
      'COMMENT:"Confronto fra gli incidenti nel quadriennio 2002-2005"
'.
      'GPRINT:media2002:"Media 2002 %4.0lf" '.
      'GPRINT:max2002:"Massimo 2002 %4.0lf" '.
      'GPRINT:mediana2002:"Mediana 2002 %4.0lf" '.
      'GPRINT:media2003:"Media 2003 %4.0lf" '.
      'GPRINT:max2003:"Massimo 2003 %4.0lf" '.
      'GPRINT:mediana2003:"Mediana 2003 %4.0lf" '.
      'GPRINT:media2004:"Media 2004 %4.0lf" '.
      'GPRINT:max2004:"Massimo 2004 %4.0lf" '.
      'GPRINT:mediana2004:"Mediana 2004 %4.0lf" '.
      'GPRINT:media2005:"Media 2005 %4.0lf     " '.
      'GPRINT:max2005:"Massimo 2005 %4.0lf     " '.
      'GPRINT:mediana2005:"Mediana 2005 %4.0lf"';

my $agent="Mozilla/5.0 (X11; U; Linux i686; it-IT; rv:1.7.12) ".
      "Gecko/20051010 Firefox/1.0.7";

#data iniziale
my $start_date='01/03/2001';
#intervallo fra due date successive
my $step_interval="+1g";
#attesa fra una acquisizione ed l'altra
my $step_wait=0;
#data ultimo passo
my $stop_date='ieri';
my $debug=1;
my $file_name    ='polstrada.csv';
my $file_header  ="Anno\tMese\tGiorno\t".
      "PattuglieAutostrada\tPattuglieSS\t".
      "IncidentiAutostrada\tIncidentiSS\t".
      "IncMortaliAutostrada\tIncMortaliSS\t".
      "IncConFeritiAutostrada\tInciConFeritiSS\t".
      "DecedutiAutostrada\tDecedutiSS\t".
      "FeritiAutostrada\tFeritiSS";
my $file_content ='"$AAAA\t$MM\t$GG\t'.
      '$dati{pattuglie_as}\t$dati{pattuglie_ss}\t'.
      '$dati{incidenti_as}\t$dati{incidenti_ss}\t'.
      '$dati{imortali_as}\t$dati{imortali_ss}\t'.
      '$dati{iferiti_as}\t$dati{iferiti_ss}\t'.
      '$dati{deceduti_as}\t$dati{deceduti_ss}\t'.
      '$dati{feriti_as}\t$dati{feriti_ss}";';

my $url = "http://www.poliziadistato.it/pds/stradale/".
      "archivio/zoom.php?vg=[GG].[MM].[AAAA]";
```

```
print "Elimino l'archivio: \n";
`rm polstrada.rrd`;
print "Ricreo l'archivio: \n";
`$rrd_create`;
open(OUTPUT, ">$file_name");
print OUTPUT "$file_header\n";
close(OUTPUT);

my $mech = WWW::Mechanize->new(
        agent=>$agent
);
my $rrd = RRDTool::OO->new(
      file => "polstrada.rrd" );
&Date_Init("Language=Italian","DateFormat=non-US");

my $date_start = ParseDate($start_date);
my $date_curr  = ParseDate($start_date);
my $date_stop  = ParseDate($stop_date);
my $date_delta = ParseDateDelta($step_interval);
my $reg_cella_piena='<td.*?>.*?<span.*?>(\d*)<\/span>.*?<\/td>.*?';
my $reg_cella_vuota='<td.*?>.*?<\/td>.*?';
my $reg_p='Totale delle pattuglie impiegate.*?<\/td>.*?'.
      $reg_cella_piena.$reg_cella_piena;
my $reg_i='Totale incidenti\s+rilevati.*?<\/td>.*?'.
      $reg_cella_piena.$reg_cella_piena;
my $reg_im='Incidenti\s+mortali.*?<\/td>.*?'.
      $reg_cella_piena.$reg_cella_piena;
my $reg_if='Incidenti\s+con\s+feriti.*?<\/td>.*?'.
      $reg_cella_piena.$reg_cella_piena;
my $reg_m='Persone\s+decedute.*?<\/td>.*?'.
      $reg_cella_piena.$reg_cella_piena;
my $reg_f='Persone ferite.*?<\/td>.*?'.
      $reg_cella_piena.$reg_cella_piena;

while (Date_Cmp($date_curr, $date_stop)<=0) {
      my $GG   =UnixDate($date_curr, '%d');
      my $MM   =UnixDate($date_curr, '%m');
      my $AAAA =UnixDate($date_curr, '%Y');
      print "Recupero i dati del giorno $GG/$MM/$AAAA\n" if ($debug);

      my $url_string=$url;
      $url_string=~s/\[GG\]/$GG/i;
      $url_string=~s/\[MM\]/$MM/i;
      $url_string=~s/\[AAAA\]/$AAAA/i;

      $mech->get($url_string);
      my $html=$mech->content();
      my %dati;

      $html =~ /$reg_p/s;
      $dati{'pattuglie_as'}=$1;
      $dati{'pattuglie_ss'}=$2;

      $html =~ /$reg_i/s;
```

```
	$dati{'incidenti_as'}=$1;
	$dati{'incidenti_ss'}=$2;

	$html =~ /$reg_im/s;
	$dati{'imortali_as'}=$1;
	$dati{'imortali_ss'}=$2;

	$html =~ /$reg_if/s;
	$dati{'iferiti_as'}=$1;
	$dati{'iferiti_ss'}=$2;

	$html =~ /$reg_m/s;
	$dati{'deceduti_as'}=$1;
	$dati{'deceduti_ss'}=$2;

	$html =~ /$reg_f/s;
	$dati{'feriti_as'}=$1;
	$dati{'feriti_ss'}=$2;

	print Dumper(\%dati) if ($debug);

	$rrd->update(time => UnixDate($date_curr, "%s"),
		values => \%dati);

	open(OUTPUT, ">>$file_name");
	my $line=eval($file_content);
	print OUTPUT "$line\n";
	close(OUTPUT);

	my $err;
	$date_curr = DateCalc($date_curr,$date_delta,\$err);
	die "\n\nErrore ($err) nell'incremento della ".
		"data [$step_interval]\n\n" if ($err);

	sleep($step_wait) if ($step_wait);
}

`$rrd_graph`;

print "\n\nFinito\n\n";
```

Appendice 4: Script utili

date2ts

Converte date dal formato testuale al numero di secondi trascorsi dal primo gennaio 1970 (epoch)

```
#!/usr/bin/env perl

use AppConfig;
use Date::Manip;

if (scalar(@ARGV)<1) {
      print "uso: date2ts [--language Italian –format non-US] DATA\n";
      exit();
}

my $config = AppConfig->new({
            CASE    => 0, #case insensitive
            CREATE  => 1,
      },
      "language|l|=s",
      "format|f|=s"
);

$config->language('Italian');
$config->format('non-US');

$config->getopt();

&Date_Init("Language=".$config->language(),
      "DateFormat=".$config->format);
$date = ParseDate($ARGV[0]);
print UnixDate($date,"%s");
```

rrd2human

Rende leggibile il risultato esposto da RRD con una operazione di fetch:

```
#!/usr/bin/env perl

use Date::Manip;

my $index=1;

while(my $row=<>) {
      chomp($row);

      if ($row=~/^[012]/) {
            my ($secs, $value)=split(/: /, $row);
            print sprintf("%04d", $index);
            print " ";
            my $date = &ParseDateString("epoch $secs");
            print &UnixDate($date,"%e/%m/%Y %H:%M:%S");
            print " --> ";
            my @values=split(/ /, $value);
            for my $item (@values) {
                  print sprintf("%8d ", $item);
            }
            print "\n";
            $index++;
      } else {
            print "$row\n";
      }
}
```

Appendice 5: WebAcquire

```
#!/usr/bin/env perl

#Lettura parametri
use AppConfig;
#Visualizzazione Help on-line
use Pod::Usage;
#Navigazione Web
use WWW::Mechanize;
#Visualizzazione delle variabili
use Data::Dumper;
#Utility per gestire le date
use Date::Manip;
#Manipolazione archivi rrd
use RRDTool::OO;

#Dichiarazione Variabili
use strict;

#variabili globali di sostituzione
my %substs;
my $date_element="%y,%Y,%G,%L,%m,%f,%b,%B,%U,%W,%j,%d,%e,%v,%a,%A,%w,%
E,%H,".
      "%k,%i,%I,%p,%M,%S,%s,%o,%Z,%z,%c,%C,%g,%D,%x,%l,%r,%R,%T,%V,%
Q,%q,%P,".
      "%O,%F,%J,%K,%n,%t,%%,%+";
my $debug=0;

sub subst_string {
      my $text=shift;
      my $date=shift;

      print "Sostituzioni per\n\t$text\n" if ($debug);
      foreach my $key (keys(%substs)) {
            my $subst=$substs{$key};
            $text =~ s/\[$key\]/$subst/ig;
      }
      foreach my $frmt (split(/,/, $date_element)) {
            my $subst=UnixDate($date, $frmt);
            $text =~ s/\[$frmt\]/$subst/g;
      }
      print "\tRisultato:\n\t$text\n" if ($debug);
      return $text;
}

#Oggetto di Configurazione
my $config = AppConfig->new({
            CASE    => 0, #case insensitive
            CREATE  => 1,
      },
```

```
      "help|!",
      "debug|d|verbose|v|!",
      "config|configfile|cfg|=s@",
      "date_language|=s",
      "date_format|=s",
      "sleepy|!",
      #Acquisizione dei dati e navigazione
      "agent|=s",
      "url|u|=s",
      "action|a|=s@",
      "subst|substitute|=s%",
      #Estrazione dei dati
      "regex_cycle|!",
      "regex|=s@",
      "variables|=s@",
      "number_italian2english|!",
      #Archiviazione RRD
      "rrd_create_string|=s",
      "rrd_recreate|!",
      "rrd_graph_string|=s",
      "rrd_file|=s",
      "rrd_variables|=s",
      "rrd_update_time|=s",
      "rrd_cumulate|!",
      #archiviazione su file
      "file_recreate|!",
      "file_name|=s",
      "file_header|=s",
      "file_content|=s",
      #Iterazione
      "date_start|=s",
      "date_stop|=s",
      "date_delta|=s",
      "sleep|=s"
);

#Inizializzazione dei parametri
#Opzioni generali
$config->help(0);
$config->debug(0);
#$config->config('');
$config->date_language('Italian');
$config->date_format('non-US');
$config->sleepy(0);
#Acquisizione dei dati e navigazione
$config->agent("Mozilla/5.0 (X11; U; Linux i686; it-IT; rv:1.7.12) ".
      "Gecko/20051010 Firefox/1.0.7");
$config->url('');
#$config->action('');
#$config->subst('');
#Estrazione dei dati
$config->regex_cycle(0);
#$config->regex('');
#$config->variables('');
$config->number_italian2english(0);
```

```
#Archiviazione RRD
#$config->rrd_create_string
$config->rrd_recreate(0);
#$config->rrd_graph_string('');
$config->rrd_file('');
$config->rrd_variables('');
$config->rrd_update_time('');
$config->rrd_cumulate(0);
#archiviazione su file
$config->file_recreate(0);
$config->file_name('');
#$config->file_header('');
$config->file_content('');
#Iterazione
$config->date_start('');
$config->date_stop('');
$config->date_delta('+1m');
$config->sleep(1);

#Recupero le opzioni a riga di comando
$config->getopt();

#Visualizzo l'help se richiesto
pod2usage(-exitstatus => 0, -verbose => 2) if ($config->help());

$debug=($config->debug()?1:0);

#Carico i file di configurazione
for (my $i=0;$i<scalar(@{$config->config()});$i++) {
        my @configs=@{$config->config()};
        my $configfile=$configs[$i];
        if (-e $configfile) {
                if (-r $configfile) {
                        print "\nCaricamento configurazione da
$configfile\n" if ($debug);
                        $config->file($configfile);
                } else {
                        print "\nFile $configfile protetto in
lettura\n" if ($debug);
                }
        } else {
                print "\nFile $configfile inesistente\n" if ($debug);
        }
}
$debug=($config->debug()?1:0);

#print "Configurazione: \n" if ($debug);
#$config->_dump() if ($debug);

#Variabili obbligatorie:
#url, regex, variables, date_start, date_stop, date_delta
die("Specificare un URL\n") if ($config->url() =~ /^\s*$/ );
die("Specificare una espressione di ricerca\n")
      if ($config->regex() =~ /^\s*$/ );
die("Specificare le variabili\n") if ($config->variables() =~ /
```

```
^\s*$/ );
die("Specificare la data iniziale\n") if ($config->date_start() =~ /
^\s*$/ );

#Carico l'hash con le "Macro" di sostituzione
%substs=%{$config->subst()};

#Carico le sostituzioni a linea di comando
for (my $i=0;$i<scalar(@ARGV);$i++) {
      my ($k, $v)=split(/=/, $ARGV[$i], 2);
      print "Sostituzione a linea di comando $k=$v\n" if ($debug);
      $substs{$k}=$v;
}

#controllo se creare l'archivio RRD
my $rrd_save=1;
my $rrd;
$rrd_save=0 if ($config->rrd_file()=~/^\s*$/);
my $rrd_update_time=$config->rrd_update_time();
my $rrd_variables=$config->rrd_variables();
$rrd_variables=",$rrd_variables," unless ($rrd_variables =~ /^\s*$/);
if ($rrd_save) {
      if ($config->rrd_recreate()==1 ||
            !(-e $config->rrd_file())) {
            my $temp=$config->rrd_create_string();
            my $result=`$temp`;
            print "Creato l'archivio RRD: risultato $result\n" if
($debug);
      }
      $rrd = RRDTool::OO->new(
            file => $config->rrd_file() );
}

#controllo se creare l'archivio di testo
my $file_name=$config->file_name();
my $file_save=1;
my $file_content=$config->file_content();
$file_save=0 if ($file_name=~/^\s*$/);
if ($file_save) {
      if ($config->file_recreate()==1 ||
            !(-e $file_name)) {
            print "Creazione fiel CSV [$file_name]\n" if ($debug);
            open(OUTPUT, ">$file_name");
            my $header=$config->file_header();
            $header=eval($header);
            print OUTPUT "$header\n";
            close(OUTPUT);
            print "Creato l'archivio nel file $file_name\n" if
($debug);
      }
}

#inizializzo mechanize, eventualmente in modalita sleepy
my $mech;
if ($config->sleepy()) {
```

```
      $mech = WWW::Mechanize::Sleepy->new(
            agent=>$config->action()
      );
} else {
      $mech = WWW::Mechanize->new(
            agent=>$config->action()
      );
}

#Inizializzo il parsing della data
&Date_Init("Language=".  $config->date_language(),
           "DateFormat=".$config->date_format());

my $date_start = ParseDate($config->date_start());
my $date_curr  = $date_start;
my $date_stop  = $date_start;
$date_stop     = ParseDate($config->date_stop()) unless ($config-
>date_stop() =~ /^\s*$/);
my $date_delta = ParseDateDelta($config->date_delta());

my @regexs=@{$config->regex()};
my @vars=@{$config->variables()};
#my @variables;
#for (my $i=0;$i<scalar(@vars);$i++) {
#     print "Linea $i: $vars[$i]\n";
#     $variables[$i]=split(/,/, $vars[$i]);
#     print "Qui";
#     my @tmp=@{\$variables[$i]};
#     print "Qua";
#     print "Esploso: ".join('--',@tmp)."\n";
      #my @v=split(/,/, $vars[$i]);
      #print "Esplodo: ";
      #for (my $j=0;$j<scalar(@v); $j++) {
      #
      #     $variables[$i][$j]=$v[$j];
      #}
#}
my $step_wait=$config->sleep();

while (Date_Cmp($date_curr, $date_stop)<=0) {
      my $url=subst_string($config->url(), $date_curr);
      print "\nMi connetto a $url\n" if ($debug);
      $mech->get($url);

      #Eseguo le azioni
      my @actions=@{$config->action()};
      for (my $i=0; $i<scalar(@actions); $i++) {
            my $action=subst_string($actions[$i], $date_curr);

            my @elements = split(/\|/, $action);
            if ($elements[0] =~ /submit_form/) {
                  my $form_number=$elements[1];
                  print "Carico il form numero $form_number. Campi: "
if ($debug);
                  $mech->form_number($form_number);
```

```
                for (my $e=2;$e<scalar(@elements);$e+=2) {
                      my $k=$elements[$e];
                      my $v=$elements[$e+1];
                      print "$k=$v " if ($debug);
                      $mech->field($k, $v);
                }
                print "\n" if ($debug);
                $mech->submit();
          } elsif ($elements[0] =~ /follow_url$/) {
                my $link=$elements[1];
                print "\nSeguo il Link $link\n" if ($debug);
                $mech->follow_link( url => $link );
          } elsif ($elements[0] =~ /follow_url_regex$/) {
                my $link=$elements[1];
                print "\nSeguo il Link $link (regex)\n" if ($debug);
                $mech->follow_link( url_regex => qr/$link/i );
          } else {
                print "\nAzione NON riconosciuta ".$elements[0]."\n";
          }
          print "\n" if ($debug);
    }

    #Recupero il risultato
    print "Acquisisco il risultato HTML\n" if ($debug);
    my $html = $mech->content();
    #print "$html\n----------------\n" if ($debug);

    if ($config->regex_cycle()) {
          my $regex=subst_string($regexs[0], $date_curr);
          my @variables=split(/,/, $vars[0]);
          my $founded=0;
          my %rrd_hash;
          while ($html =~ /$regex/sg) {
                my %dati;
                my $j=1;
                foreach my $tmp (@variables) {
                      my $tmp2=eval('$'.$j);
                      print "variabile $tmp, valore $tmp2\n";
                      $dati{$tmp}=$tmp2;
                      $j++;
                }

                if ($config->number_italian2english()) {
                      foreach my $tmp (keys(%dati)) {
                            $dati{$tmp}=~s/\.//g;
                            $dati{$tmp}=~s/,/./g;
                      }
                }

                if ($file_save) {
                      open(OUTPUT, ">>$file_name");
                      my $line=subst_string($file_content,
$date_curr);
                      my $line=eval($line);
```

```
                        print OUTPUT "$line\n";
                        close(OUTPUT);
                  }

                  my $tmp_time=$date_curr;
                  unless ($rrd_update_time =~ /^\s*$/) {
                        my $tmp='"'.subst_string($rrd_update_time,
$date_curr).'"';
                        $tmp=eval($tmp);
                        $tmp_time=ParseDate($tmp);
                        print "Data Corrente: ".UnixDate($date_curr, "%
s")."\n".
                              "   modificata: ".UnixDate($tmp_time,
"%s")."\n"
                              if ($debug);
                  }
                  unless ($rrd_variables =~ /^\s*$/) {
                        foreach my $tmp (keys(%dati)) {
                              unless ($rrd_variables =~ /,$tmp,/) {
                                    delete $dati{$tmp};
                                    print "Elimino $tmp dal salvataggio
in RRD\n";
                              }
                        }
                  }
                  if ($config->rrd_cumulate()) {
                        $rrd_hash{'_'.UnixDate($tmp_time, "%s")} = \%
dati;
                  } else {
                        $rrd->update(time => UnixDate($tmp_time, "%s"),
                              values => \%dati) if ($rrd_save);
                  }
                  $founded=1;
            }
            if ($founded==0) {
                  print "NESSUNA Espressione Regolare non trovata\n" if
($debug);
            } elsif ($config->rrd_cumulate()) {
                  foreach my $tmp (sort(keys(%rrd_hash))) {
                        $tmp =~ s/^_//;
                        $rrd->update(time => $tmp,
                              values => $rrd_hash{"_$tmp"}) if
($rrd_save);
                  }
            }
      } else {
            my %dati;
            my $founded=0;
            for (my $i=0;$i<scalar(@regexs);$i++) {
                  #my @tmp=split(/,/, $vars[$i]);
                  my $regex=subst_string($regexs[$i], $date_curr);

                  #il modificatore s permettedi considerare il
documento
                  #come su una unica riga (per far si che .* comprenda
```

```
anche gli
                  #a-capo, cosiccome \s
                  my @variables=split(/,/, $vars[$i]);
                  if ($html =~ /$regex/s) {
                        my $j=1;
                        foreach my $tmp (@variables) {
                              my $tmp2=eval('$'.$j);
                              print "variabile $tmp, valore $tmp2\n";
                              $dati{$tmp}=$tmp2;
                              $j++;
                        }
                        $founded=1;
                  } else {
                        print "Espressione $regex non trovata\n" if
($debug);
                  }
            }

            if ($founded) {
                  if ($config->number_italian2english()) {
                        foreach my $tmp (keys(%dati)) {
                              $dati{$tmp}=~s/\.//g;
                              $dati{$tmp}=~s/,/./g;
                        }
                  }

                  if ($file_save) {
                        open(OUTPUT, ">>$file_name");
                        my $line=subst_string($file_content,
$date_curr);
                        my $line=eval($line);
                        print OUTPUT "$line\n";
                        close(OUTPUT);
                  }

                  my $tmp_time=$date_curr;
                  unless ($rrd_update_time =~ /^\s*$/) {
                        my $tmp='"'.subst_string($rrd_update_time,
$date_curr).'"';
                        $tmp=eval($tmp);
                        $tmp_time=ParseDate($tmp);
                        print "Data Corrente: ".UnixDate($date_curr, "%
s")."\n".
                              "   modificata: ".UnixDate($tmp_time,
"%s")."\n"
                              if ($debug);
                  }
                  unless ($rrd_variables =~ /^\s*$/) {
                        foreach my $tmp (keys(%dati)) {
                              unless ($rrd_variables =~ /,$tmp,/) {
                                    delete $dati{$tmp};
                                    print "Elimino $tmp dal salvataggio
in RRD\n";
                              }
                        }
```

```
                  }
                  $rrd->update(time => UnixDate($date_curr, "%s"),
                        values => \%dati) if ($rrd_save);
            } else {
                  print "NESSUNA Espressione Regolare non trovata\n" if
($debug);
            }
      }

      my $err;
      $date_curr = DateCalc($date_curr,$date_delta,\$err);
      die "\n\nErrore ($err) nell'incremento della ".
            "data [".$config->date_delta()."]\n\n" if ($err);

      sleep($step_wait) if ($step_wait);
}

my $rrd_graph=$config->rrd_graph_string();
unless ($rrd_graph=~/^\s*$/) {
      my $result=`$rrd_graph`;
      print "Creato il grafico RRD [$rrd_graph]: risultato $result\n"
if ($debug);
}

print "\nFinito.\n\n" if ($debug);

=head1 NAME

webacquire - Acquisisce contenuti dal web eseguendo semplici
navigazioni

=head1 SYNOPSYS

webacquire --url URL [--config CONFIGFILE] [--agent AGENTSTRING]
      [[--action ACTION1] ...]
      [[--search SEARCH1] ...] [[--replace REPLACE11] ...]
      [[--subst KEY=VALUE] ...]
      [--debug] [--help] PARAMETRO=VALORE ...
Options:
  --debug, --verbose       attivano la modalit‡ "verbose"
  --help                   visualizza questo help
  --agent AGENTSTRING      stringa "agent" simulata dal browser
  --action ACTION1 ...     elenco delle azioni da compiere
(follow_url, follow_url_regex, submit_form, ...)
  --search SEARCH1 ...     stringa da ricercare nel risultato
  --replace REPLACE11 ...  stringa da sostituire nel risultato
  --subst KEY=VALUE ...    espande nelle azioni il testo [KEY] con il
valore [VALUE] (solo nei file di configurazione)
  PARAMETRO=VALORE         equivalente a --subst KEY=VALUE a linea di
comando

I parametri contrassegnati da (...) possono essere anche multipli.

Le opzioni nel file di configurazione vengono sovrascritte o integrate
con quelle a linea di comando
```

```
Esempio:
      webacquire --url "http://www.google.com" --action "submit_form|
0|q|e-mbuto"

=head1 OPTIONS

=over 8

=item B<--debug>
=item B<--verbose>
Attiva la modalit‡ verbosa
=item B<--config>
Specifica il file di configurazione, anche pi˘ di uno (ad esempio --
config=/etc/conf1.conf --config=/etc/conf2.conf)

=back

=head1 DESCRIPTION

Acquisisce contenuti dal web eseguendo semplici navigazioni,
riempiendo form, aprendo frame e seguendo i link

=cut
```

Appendice 6: Configurazioni di WebAcquire

Borsa Italiana

```
rrd_create_string='rrdtool create borsa.rrd \
      --step 60 \
      --start `./date2ts oggi` \
      DS:valore:GAUGE:`echo 60*20 | bc`:0:50000 \
      RRA:AVERAGE:0.5:1:`echo 24*365*2 | bc` \
      RRA:AVERAGE:0.5:15:40 \
      RRA:AVERAGE:0.5:60:`echo 24*7*30 | bc`'

rrd_graph_string='rrdtool graph borsa.png \
      --end `./date2ts "28/05/2006 20:35"` --start end-1h --step 60 \
      DEF:valore=borsa.rrd:valore:AVERAGE:\
end=`./date2ts "28/05/2006 20:35"`:start=end-1h \
      LINE1:valore#00ff00:"Andamento di ieri"'

rrd_recreate=off
rrd_file=borsa.rrd
rrd_variables='valore'
rrd_update_time='[%d]/[%m]/[%Y],$dati{ore}:$dati{minuti}:00'
rrd_cumulate=on

file_name='borsa.csv'
file_recreate=off
file_header='"Anno\tMese\tGiorno\tOra\tMinuti\tValore"'
file_content='"[%Y]\t[%m]\t[%d]\t$dati{ore}\t$dati{minuti}\t$dati
{valore}"'

date_start='adesso'
date_stop='oggi,20:30'
#date_stop=''
date_delta='+5minuti'

sleep=0
debug=on
number_italian2english=on

url =
"http://www.borsaitalia.it/bitApp/intraday.bit?isin=II935&target=indic
i&lang=it&from=[%H][%M]&querymode=byTime&HH=[%H]&MM=[%M]"

regex_cycle=on
regex='<\!\-\- IF class=tdazz\/tdbia \-\->\s*<tr class\="(tdazz|
```

```
tdbia)">\s*<td class\="dato">(\d{2})\.(\d{2})\.00<\/td>\s*<td
class\="dato">([0-9.,]*)</td>\s*<\!\-\- IF class=red\/green\/blu \-\->
\s*<td class\="(red|green|blu)">([0-9.,\-\+]*)<\/td>\s*<td
class\="dato">([0-9]*)<\/td>\s*</tr>'
variables='temp1,ore,minuti,valore'
```

Polizia Stradale

```
rrd_create_string="rrdtool create polstrada.rrd \
      --step `echo 60*60*24 | bc` \
      --start `./date2ts 28/02/2001` \
      DS:pattuglie_as:GAUGE:172800:0:5000 \
      DS:incidenti_as:GAUGE:172800:0:1000 \
      DS:imortali_as:GAUGE:172800:0:1000 \
      DS:iferiti_as:GAUGE:172800:0:1000 \
      DS:deceduti_as:GAUGE:172800:0:1000 \
      DS:feriti_as:GAUGE:172800:0:1000 \
      DS:pattuglie_ss:GAUGE:172800:0:5000 \
      DS:incidenti_ss:GAUGE:172800:0:1000 \
      DS:imortali_ss:GAUGE:172800:0:1000 \
      DS:iferiti_ss:GAUGE:172800:0:1000 \
      DS:deceduti_ss:GAUGE:172800:0:1000 \
      DS:feriti_ss:GAUGE:172800:0:1000 \
      RRA:AVERAGE:0.5:1:3650 \
      RRA:AVERAGE:0.5:1:28 \
      RRA:AVERAGE:0.5:1:90 \
      RRA:AVERAGE:0.5:7:52 \
      RRA:AVERAGE:0.5:7:150"

rrd_graph_string='rrdtool graph polstrada.png \
      --end `./date2ts 01/01/2006` \
      --start end-1y --step `echo 60*60*24*7|bc` \
      DEF:incidenti2002=polstrada.rrd:incidenti_ss:AVERAGE:\
end=`./date2ts 01/01/2003`:start=end-1y \
      DEF:incidenti2003=polstrada.rrd:incidenti_ss:AVERAGE:\
end=`./date2ts 01/01/2004`:start=end-1y \
      DEF:incidenti2004=polstrada.rrd:incidenti_ss:AVERAGE:\
end=`./date2ts 01/01/2005`:start=end-1y \
      DEF:incidenti2005=polstrada.rrd:incidenti_ss:AVERAGE:\
end=`./date2ts 01/01/2006`:start=end-1y \
      SHIFT:incidenti2004:`echo 60*60*24*365   | bc` \
      SHIFT:incidenti2003:`echo 60*60*24*365*2 | bc` \
      SHIFT:incidenti2002:`echo 60*60*24*365*3 | bc` \
      VDEF:media2002=incidenti2002,AVERAGE \
      VDEF:max2002=incidenti2002,MAXIMUM \
      VDEF:mediana2002=incidenti2002,50,PERCENT \
      VDEF:media2003=incidenti2003,AVERAGE \
```

```
      VDEF:max2003=incidenti2003,MAXIMUM \
      VDEF:mediana2003=incidenti2003,50,PERCENT \
      VDEF:media2004=incidenti2004,AVERAGE \
      VDEF:max2004=incidenti2004,MAXIMUM \
      VDEF:mediana2004=incidenti2004,50,PERCENT \
      VDEF:media2005=incidenti2005,AVERAGE \
      VDEF:max2005=incidenti2005,MAXIMUM \
      VDEF:mediana2005=incidenti2005,50,PERCENT \
      LINE1:incidenti2002#0000ff:"Incidenti anno 2002" \
      LINE1:incidenti2003#00ff00:"Incidenti anno 2003" \
      LINE1:incidenti2004#ff0000:"Incidenti anno 2004" \
      LINE1:incidenti2005#121212:"Incidenti anno 2005" \
      COMMENT:"Confronto fra gli incidenti nel periodo 2002-2005" \
      GPRINT:media2002:"Media 2002 %4.0lf" \
      GPRINT:max2002:"Massimo 2002 %4.0lf" \
      GPRINT:mediana2002:"Mediana 2002 %4.0lf" \
      GPRINT:media2003:"Media 2003 %4.0lf" \
      GPRINT:max2003:"Massimo 2003 %4.0lf" \
      GPRINT:mediana2003:"Mediana 2003 %4.0lf" \
      GPRINT:media2004:"Media 2004 %4.0lf" \
      GPRINT:max2004:"Massimo 2004 %4.0lf" \
      GPRINT:mediana2004:"Mediana 2004 %4.0lf" \
      GPRINT:media2005:"Media 2005 %4.0lf     " \
      GPRINT:max2005:"Massimo 2005 %4.0lf     " \
      GPRINT:mediana2005:"Mediana 2005 %4.0lf"'

rrd_recreate=on
rrd_file=polstrada.rrd

date_start='01/03/2001'
date_stop='ieri'
date_delta='+1g'

sleep=5
debug=on

file_name='polstrada.csv'
file_recreate=on
file_header='"Anno\tMese\tGiorno\tPattuglieAutostrada\tPattuglieSS\tIn
cidentiAutostrada\tIncidentiSS\tIncMortaliAutostrada\tIncMortaliSS\tIn
cConFeritiAutostrada\tInciConFeritiSS\tDecedutiAutostrada\tDecedutiSS\
tFeritiAutostrada\tFeritiSS"'

file_content='"[%Y]\t[%m]\t[%d]\t$dati{pattuglie_as}\t$dati
{pattuglie_ss}\t$dati{incidenti_as}\t$dati{incidenti_ss}\t$dati
{imortali_as}\t$dati{imortali_ss}\t$dati{iferiti_as}\t$dati
{iferiti_ss}\t$dati{deceduti_as}\t$dati{deceduti_ss}\t$dati{feriti_as}
\t$dati{feriti_ss}"'

url = "http://www.poliziadistato.it/pds/stradale/archivio/zoom.php?vg=
[%d].[%m].[%Y]"

subst reg_cella_piena='<td.*?>.*?<span.*?>(\d*)<\/span>.*?<\/td>.*?'
subst reg_cella_vuota='<td.*?>.*?<\/td>.*?'
```

```
regex='Totale delle pattuglie impiegate.*?<\/td>.*?[reg_cella_piena]
[reg_cella_piena]'
variables='pattuglie_as,pattuglie_ss'

regex='Totale incidenti\s+rilevati.*?<\/td>.*?[reg_cella_piena]
[reg_cella_piena]'
variables='incidenti_as,incidenti_ss'

regex='Incidenti\s+mortali.*?<\/td>.*?[reg_cella_piena]
[reg_cella_piena]'
variables='imortali_as,imortali_ss'

regex='Incidenti\s+con\s+feriti.*?<\/td>.*?[reg_cella_piena]
[reg_cella_piena]'
variables='iferiti_as,iferiti_ss'

regex='Persone\s+decedute.*?<\/td>.*?[reg_cella_piena]
[reg_cella_piena]'
variables='deceduti_as,deceduti_ss'

regex='Persone ferite.*?<\/td>.*?[reg_cella_piena][reg_cella_piena]'
variables='feriti_as,feriti_ss'
```

ARPA Emilia Romagna

```
file_name='arpa.csv'
file_recreate=off
file_header='"Anno\tMese\tGiorno\tCarpi\tFiorano\tModena1\tModena2\tMo
dena3"'
file_content='"$dati{anno}\t$dati{mese}\t$dati{giorno}\t$dati{fiorano}
\t$dati{carpi}\t$dati{modena1}\t$dati{modena2}\t$dati{modena3}"'

date_start='adesso'
date_stop=''

sleep=0
debug=on

url = "http://service.arpa.emr.it/qualita-aria-
2005/bollettino.aspx?prov=mo"

#recupera il giorno
regex='<option selected="selected" value="(\d\d)\/(\d\d)\/(\d\d\d\d)"'
variables='giorno,mese,anno'

#recupera i valori
```

```
regex='CARPI \- CARPI 2 \(VIA REMESINA \)<\/td><td>(.*?)<\/td>'
variables='carpi'
regex='FIORANO MODENESE \- SPEZZANO 2 \(VIA MOLINO \)<\/td><td>(.*?)
<\/td>'
variables='fiorano'
regex='MODENA \- MO \- PARCO FERRARI \( PARCO FERRARI \)<\/td><td>
(.*?)<\/td>'
variables='modena1'
regex='MODENA \- MO \- VIA GIARDINI \(VIA GIARDINI \)<\/td><td>(.*?)
<\/td>'
variables='modena2'
regex='MODENA \- MO \- VIA NONANTOLANA \(VIA CIMONE \)<\/td><td>(.*?)
<\/td>'
variables='modena3'
```

Bibliografia

Alan Seiden, "Use the i5's PASE environment to run cURL",
Mc Press, Maggio 2006

Adrian J. Chung, "Downloading without a Browser",
Linux Gazette, Settembre 2001

Ethan McCallum, "Simplify Network Programming with libCUR",
Linux DevCenter, O'Really, Maggio 2005

James Howison, "Studying Free Software with Free Software and Free method", Australian Open Source Development Conference, Dicembre 2004

Davide Eynard, "Powerbrowsing: un robot per la navigazione web",
Linux & C. numero 52, Maggio 2006

Simon St. Laurent, Joe Jhonston, Edd Dumbill, "Programmare Web Services XML-RPC", Hops Libri, 2002

David Cross, "Data Munging With Perl",
Manning, 2001

Simon Cozens, "Advanced Perl Programming 2nd Edition",
O'Reilly, 2005

Larry Wall, "Programming Perl",
O' Really, 2000

T. Fioreze, L. Z. Granville, M. J. Almeida, L. R. Tarouco, "Comparing Web Services with SNMP in a Management by Delegation Environment", Federal University of Rio Grande do Sul, 2005

Dominique Lalot, "MRTG est mort vive RRDTOOL",
CISCAM Université de la Méditerranée, 2003,
http://2003.jres.org/actes/paper.29.pdf

Steve Reader, "RPN Tutorial",
http://oss.oetiker.ch/rrdtool/tut/rpntutorial.en.html
Alex van den Bogaerdt, "CDEF Tutorial",

http://oss.oetiker.ch/rrdtool/tut/cdeftutorial.en.html

Alex van den Bogaerdt, "RRD Tutorial",
http://oss.oetiker.ch/rrdtool/tut/rrdtutorial.en.html

ketan Petel, "RRD Beginners",
http://oss.oetiker.ch/rrdtool/tut/rrd-beginners.en.html

AAVV, "Manuale di WGET",
http://www.gnu.org/software/wget/manual

Michael Bolin, "End-User Programming for the Web",
MASSACHUSETTS INSTITUTE OF TECHNOLOGY,
Maggio 2005