ISSN 2039-7941
Anno 2017_Numero 375

rapporti tecnici INGV

Strumento di Diagnostica e Amministrazione Remota di reti per il monitoraggio di parametrici geofisici

375

Istituto Nazionale di Geofisica e Vulcanologia

Segreteria di Redazione
Francesca Di Stefano - Referente
Rossella Celi
Tel. +39 06 51860068
redazionecen@ingv.it

in collaborazione con:
Barbara Angioni (RM1)

REGISTRAZIONE AL TRIBUNALE DI ROMA N.173 | 2014, 23 LUGLIO

Rappresentante legale: Carlo DOGLIONI
Sede: Via di Vigna Murata, 605 | Roma

ISSN 2039-7941
Anno 2017_Numero 375

rapporti tecnici

INGV

STRUMENTO DI DIAGNOSTICA E AMMINISTRAZIONE REMOTA DI RETI PER IL MONITORAGGIO DI PARAMETRICI GEOFISICI

Antonino Sicali, Pasqualino Cappuccio, Alfio Amantia

INGV (Istituto Nazionale di Geofisica e Vulcanologia, Sezione di Catania - Osservatorio Etneo)

375

Come citare: Sicali A., Cappuccio P. e Amantia A., (2017). Strumento di Diagnostica e Amministrazione Remota di reti per il monitoraggio di parametrici geofisici. Rapp. Tec. INGV, 375: 1-78.

Indice

Introduzione 7

1. Importanza della diagnostica per una manutenzione efficace 7
2. Amministrazione dei sistemi 8
3. Importanza dei tempi d'intervento 8
4. Descrizione del sistema 9
   - 4.1 Utilizzo dei messaggi per veicolare le informazioni 10
   - 4.2 Sicurezza e filosofia del modello adottato 11
   - 4.3 Rapidità di accesso alle informazioni 12
   - 4.4 Riconoscimento dei pattern sui messaggi 14
   - 4.5 Attuatori per l'amministrazione remota dei sistemi 15
   - 4.6 Funzionamento asincrono grazie all'invio delle email 15
5. Analisi dei segnale per estrarre informazioni utili alla diagnostica 16
   - 5.1 Analisi della scarica notturna delle batterie 16
   - 5.2 Analisi di correlazione minimo/massimo giornaliero 18
   - 5.3 Analisi del segnale principale (analisi delle maree) 20
6. Applicazione *Android* 22
7. Installazione e utilizzo 24
   - 7.1 Installazione e descrizione modulo centrale FastCGI 25
   - 7.2 Installazione e descrizione clients diagnostici 27
   - 7.3 Installazione e descrizione clients amministrativi 27

Conclusioni 28

Ringraziamenti 28

Bibliografia 28

Sitografia 29

Appendice A1. Codice sorgente: web.admin.h 30

Appendice A2. Codice sorgente: web.admin.cpp 33

Appendice A3. Codice sorgente: Makefile web.admin 59

Appendice A4. Codice sorgente: web.admin.css 60

Appendice A5. Codice sorgente: delay.h 67

Appendice A6. Codice sorgente: delay.cpp 68

Appendice A7. Codice sorgente: rdat.reset.h 70

Appendice A8. Codice sorgente: rdat.reset.cpp 71

Appendice A9. Codice sorgente: rdat.reset.sh 74

Appendice A10. Codice sorgente: Makefile rdat.reset 75

Introduzione

La manutenzione dei sistemi di monitoraggio remoto è importante poiché permette a un sistema di continuare a funzionare in modo continuo e duraturo. La stabilità dei sistemi dipende dal tipo di manutenzione che si esegue [Sicali et al., 2016]. La manutenzione copre diversi aspetti, dalla cura delle infrastrutture fino alla consistenza del software, passando per il benessere dell'hardware (elettronica). La cura delle infrastrutture deve essere effettuata in sito, per lo più realizzata come manutenzione ordinaria programmabile. Riguarda tutto ciò che risente delle intemperie e che è sottoposta a *stress* meccanici, di temperatura, ossidazione e corrosione. La manutenzione del software e dell'hardware, normalmente, viene effettuata di fronte a un problema già conclamato, che non si è potuto prevedere e/o programmare. Lo scopo del sistema esaminato di seguito è quello di aiutare, in una qualche misura, a prevedere i malfunzionamenti prima che si verifichino attraverso il constante monitoraggio di particolari segnali. Ciò che negli hard disk è stato realizzato attraverso l'adozione dei sistemi S.M.A.R.T [Wikipedia, Self-Monitoring, Analysis and Reporting Technology].

1. Importanza della diagnostica per una manutenzione efficace

Il sistema *Mag-Net* [Del Negro et al., 2002] nonostante sia il più evoluto tra quelli impiegati nelle reti magnetica, dilatometrica e gravimetrica del M.te Etna e dell'isola di Stromboli [Sicali et al., 2016], possiede un sistema di diagnostica troppo limitato. Le informazioni sui malfunzionamenti non vengono trasferite al sistema di controllo centrale per un'attenta analisi. Quando si è progettato il sistema *Mag-Net* fu scelta una strada che prevedeva la totale risoluzione automatica per ogni problema che insorgeva. Ciò nonostante negli anni si è visto che molti problemi non possono comunque essere risolti in automatico ma necessitato dell'intervento umano. Questo perché inizialmente si sono considerati solo problemi di tipo software e transitori. Nella realtà i problemi che affliggono un sistema di acquisizione possono andare al di là del semplice *bug* di programmazione. In passato si è preferito, di fronte a un malfunzionamento, semplicemente riavviare il sistema supponendo che il problema fosse generato da una struttura dati volatile corrotta. Nella realtà l'errore potrebbe derivare da problemi non transitori: un'anomalia hardware, una struttura dati sul disco inconsistente o un dispositivo esterno in avaria. Si rischia invece di perdere il collegamento telemetrico, tra il sistema e il centro di controllo, a causa dei continui riavvii, che non risolvono in modo definitivo il problema. Una strategia migliore prevede invece che il software salvaguardi il collegamento telemetrico per trasferire quante più informazioni verso il centro di controllo. La risoluzione dei problemi in automatico dovrebbe essere molto mirata e locale a ciascun modulo software o hardware [Sicali et al., 2016]. Il sistema centrale sarà riavviato solo in condizioni di reale necessità. Ricercando i segnali corretti nel funzionamento dei sistemi di acquisizione, si è visto anche che molti problemi si possono nella corretta misura prevedere. Gli strumenti diagnostici, da semplici informazioni decisionali, dovrebbero evolversi ed espandersi, per coprire ciascun aspetto sia software che hardware, per fornire una quantità enorme d'informazione sul funzionamento del sistema. Tali informazioni potranno servire, localmente per risolvere i problemi banali, o rielaborate nel centro di controllo, per prevedere con precisione eventuali malfunzionamenti futuri. Informazioni sul sistema energetico potranno essere usate per evidenziare problemi alle batterie, ai pannelli solari e al loro regolatore. Allo stesso modo informazioni sul sistema di trasmissione potranno essere usate per prevedere un cambiamento in atto della qualità del segnale, problemi alle antenne e di altri apparati eventualmente presenti.

Sembra banale costruire un tale sistema ma nella realtà non tutti i dispositivi generano segnali diagnostici o potrebbero essere riprogrammati per farlo. Spesso un dispositivo hardware esterno è completamente chiuso (*blackbox*) per cui è impossibile forzarlo a produrre qualsiasi segnale di tipo diagnostico, anche semplice. Per questo, di fronte a una necessità tecnologica conviene impiegare dispositivi aperti o progettarne di nuovi. Un dispositivo di cui si conosce il funzionamento sarà, sul lungo periodo, molto più affidabile della più affidale *blackbox*.

I sistemi di diagnostica raccolgono, grazie a una serie di test o misure, informazioni utili sullo stato di un dispositivo hardware o di un sistema software. L'insieme delle informazioni raccolte deve permette d'isolare, senza ambiguità, particolari problemi. Per ogni singolo dispositivo, modulo, elemento o struttura dati dovranno essere studiati un insieme di test e misure, per mettere in risalto i problemi del sistema, anche ancora non conclamati. Devono essere raccolte informazioni essenziali, non ridonanti, in numero molto elevato, che permetteranno d'identificare con precisione ogni anomalia del sistema e soprattutto d'intervenire nei tempi opportuni.

2. Amministrazione dei sistemi

Il passo successivo alla diagnostica è la manutenzione, che non riguarda solamente la sostituzione dell'elemento danneggiato ma anche i modi e i tempi con cui sono sostituiti o riparati. La velocità e l'efficienza/efficacia degli interventi sono parametri fondamentali, che dipendono soprattutto dalla robustezza del sistema diagnostico e dal tempestivo allarme generato. Gli automatismi che utilizzano le informazioni diagnostiche come d'ingresso, permettono di ridurre i tempi d'intervento. Riducono al minimo gli interventi di manutenzione manuale, allungando il tempo medio tra un intervento fisico e l'altro, ovvero massimizzano il parametro MTBM (Mean Time Between Maintenance). Questo perché i malfunzionamenti risolti completamente in automatico evitano che interventi invasivi possano, per distrazione e/o stanchezza dell'operatore, danneggiare la strumentazione causando ulteriori problemi, anche non immediatamente visibili. Accade molto spesso che interventi di manutenzione, se eseguiti da personale non altamente qualificato, comportano un rischio reale per il buon funzionamento di un sistema. Conviene, quindi, eseguire i test diagnostici completamente in automatico, preventivamente e distribuendoli nel tempo. Anche per le procedure di amministrazione e risoluzione dei problemi non completamente automatizzabili e particolarmente complesse e articolate, conveniente studiare un meccanismo semi automatico supervisionato dall'utente per ridurre i costi e i tempi d'intervento. In generale l'automazione delle procedure di amministrazione sarà possibile solo se viene fornito un quadro diagnostico molto preciso e curato, che permettere l'individuazione e la risoluzione anche dei problemi più nascosti e subdoli. Questo si ottiene se il sistema di amministrazione possiede una visione globale rispetto a tutta la catena di acquisizione, dalla generazione del dato fino al suo utilizzo.

3. Importanza dei tempi d'intervento

Quando si presenta un problema lungo la catena di acquisizione, sono molti i fattori che influenzano la tempistica con cui vengono ripristinate le funzionalità. Può essere che il problema inizialmente non si manifesti neanche con continuità. Altre volte si genera così improvvisamente che sembrano mancare veri e propri sintomi. Comunque inizi, la quantità d'informazione che si riesce a recuperare dopo il manifestarsi di un problema influenza significamente la precisione e il tempo con cui si interviene. L'ideale è poter mantenere in vita sempre un *core* minimale che garantisca la manutenzione remota di fronte a un problema anche grave. In mancanza di un *core* possono comunque aiutare le informazioni diagnostiche memorizzate localmente, se molto accurate e ben distribuite in tutti i sistemi. Ciò garantisce anche in assenza di un collegamento telemetrico, di descrivere discretamente il problema verificatosi. In presenza di collegamento telemetrico è bene che tutte le informazioni raccolte lungo la catena di acquisizione fluiscano in un unico luogo, per essere rielaborate e presentate nel modo più corretto. A seconda dell'informazione trattata può essere migliore una rappresentazione visiva (grafico) o testuale. Alcune informazione potrebbero non avere un unico significato e bisognerà incrociarle con altre.

Le operazioni che portano all'identificazione di un problema, anche se banali, solitamente sono in numero molto elevato. Se eseguite manualmente faranno perdere molto tempo ad un eventuale amministratore che vorrebbe intervenire. Per fare solo un esempio, quando ci si trova di fronte a un problema, per diagnosticarlo in manuale, bisogna collegarsi in remoto almeno a due o tre calcolatori. Recuperare informazioni da file, decodificarle se non sono facilmente comprensibili o se richiedono una qualche rappresentazione, ed infine incrociarle per estrarne il reale significato. Per eseguire tutte queste operazioni possono essere spese molte ore di lavoro. Con un sistema diagnostico tarato sui problemi più comuni si riesce a ottenere in modo automatico una diagnosi in qualche frazione di secondo. Un sistema automatizzato e centralizzato si occuperà, ogni volta che arriva una nuova informazione diagnostica, di aggiornare lo stato di funzionamento e di formulare una possibile diagnosi, consultabile immediatamente. Il sistema diagnostico centrale non farà altro che ripetere tutte le operazioni che normalmente vengono eseguite dall'amministratore, evitando errori e risparmiando tempo prezioso. Inoltre per quei casi in cui è disponibile una soluzione automatica al problema è possibile applicata. Questo è il caso del riavvio di calcolatori o di dispositivi in genere, come i modem, entrati in stallo. Utilizzare una procedura diagnostica in manuale, complessa o semplice che sia, rispetto ad una automatica farà la differenza quando ci si trova fuori sede e non si ha fisicamente accesso ai sistemi, se non con linee di comunicazione molto disturbate. Non sempre si ha la capacità d'intervenire in remoto, soprattutto se la procedura d'intervento è molto articolata e complessa. Si può invece sempre intervenire da remoto automatizzando le procedure e riducendole a un semplice processo decisionale. Questo perché è più semplice rispondere a una domanda o schiacciare un tasto da remoto che scrivere complessi comandi collegati a un terminale.

4. Descrizione del sistema

Per molti anni la diagnosi dei problemi è stata eseguita completamente in manuale grazie ai diversi *file* di *logging*. L'amministratore del sistema, che era anche la persona di maggiore esperienza per eseguire la manutenzione, si occupava di raccogliere informazioni sui dispositivi allo scopo d'isolare la fonte del malfunzionamento. Ciò comportava un enorme spreco di tempo e la necessità che le operazioni fossero compiute da personale esperto per non creare ulteriori danni ai sistemi. Inoltre alcune volte, a causa di un quadro incompleto, la diagnosi poteva risultare inesatta e imprecisa. Ciò perché alcune volte è impossibile reperire informazioni essenziali, a causa dei tempi ristretti o poiché non è proprio presente quel tipo di dato. Negli anni si è sentita sempre più la necessità di creare uno strumento che riesca ad accumulare tutte le informazioni per un utilizzo veloce ed efficiente. Il nuovo strumento sarebbe risultato perfetto se fosse stato capace di ripercorre tutti i passaggi, almeno quelli conosciuti, che normalmente esegue l'amministratore del sistema per arrivare a isolare la sorgente del problema. Un tale strumento permetterebbe indubbiamente di aumentare la velocità e la precisione con cui viene effettuata la diagnosi dei problemi che affliggono i sistemi di monitoraggio. Lo studio del problema ha portato all'identificazione di un modello software perfetto per lo scopo, molto flessibile e universale. Il modello adottato è capace d'interfacciarsi con la totalità dei sistemi in circolazione poiché sfrutta il protocollo *HTTP* e le più collaudate tecnologie di programmazione *server side*. Viene preservata allo stesso tempo l'accessibilità e la sicurezza grazie a una topologia centro stella in cui i diversi *client* non si conoscono e il server centrale esegue la raccolta esclusiva e il successivo smistamento delle informazioni. Il modello impiegato permette grazie l'uso della tecnologia *FastCGI* [Wikipedia, FastCGI] di scambiare messaggi, in piena sicurezza, con qualsiasi sistema possa leggere una normale pagina *web*. La tecnologia utilizzata permette inoltre di monitorare da qualsiasi luogo, grazie alla globalità della rete Internet, i sistemi e d'interagire, dove è possibile, con loro. Grazie al sito *web*, messo a disposizione, è possibile avere in ogni instante un quadro aggiornato dello stato di salute della rete di monitoraggio.

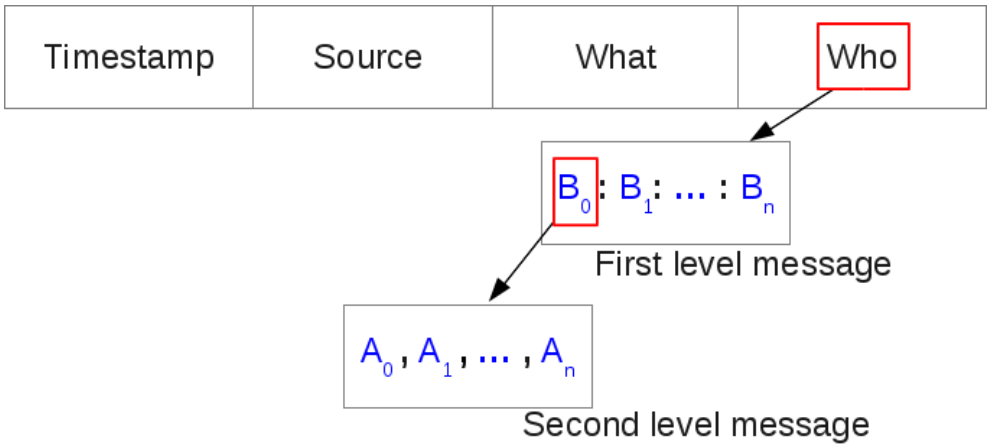

Figura 1. Struttura dei messaggi scambiati con il server per veicolare le informazioni di diagnostica.

Il sistema utilizza un modello *client-server* per raggiungere i sui obbiettivi. Semplicemente esiste un server accessibile attraverso un'interfaccia *web* che raccoglie e restituisce informazioni ai tanti moduli, i *client*s, distribuiti nei vari sottosistemi. Di *clients* ne esistono di due tipologie: i sensori e gli attuatori. Entrambe le tipologie comunicheranno e riceveranno l'informazione attraverso il metodo di richiesta *GET* del protocollo *HTTP* [Wikipedia, Hypertext Transfer Protocol]. Le operazioni di scrittura associate agli attuatori sono state trasformate in operazioni di lettura eliminando del tutto i problemi di sicurezza associati. Non è necessario bucare *firewall* o mantenere complessi canali di comunicazione per far funzionare la parte amministrativa. Ciò è stato possibile pagando un prezzo enorme, ovvero l'inserimento di un processo sincrono (polling), l'unico di tutto il sistema. I processi sincroni sono molto critici poiché possono bloccare il normale funzionamento dei sistemi e il loro utilizzo deve essere quanto più evitato [Sicali et al., 2016]. Si deve cercare di favorire maggiormente le operazioni di tipo asincrono per evitare soprattutto di sprecare inutilmente risorse. Un sistema inefficiente non scala bene verso l'alto [Wikipedia, Scalability] poiché gli sprechi fanno sentire il loro peso.

Messaggio	Formato	Descrizione
download_delay	NAME:DELAY:[YEAR][MONTH][DAY]	Comunica il ritardo accumulato durante la procedura di scaricamento dei dati
link	NAME:R_0,R_1,R_2,R_3,R_4	Comunica il risultato di test effettuati sul collegamento di rete con il sito remoto. Sono presenti due livelli di dati come mostrato in figura 1

Tabella 1. Esempi di formati di messaggi scambiati con il server.

Date	Time	Source	What	Who
13/03/17	07:10:25	ctdyndns	link	druv:256,256,256,256,256
13/03/17	07:10:25	ctdyndns	ip	druv:2.45.64.94:20170313081018
13/03/17	07:10:09	ctdyndns	signal	druv:-104_dBm.:20170313080958
13/03/17	07:07:37	ctdyndns	csq	26,99
13/03/17	07:07:37	ctdyndns	creg	0,1
13/03/17	07:07:37	ctdyndns	cops	0,0,vodafone
13/03/17	07:07:37	ctdyndns	ddns	idle
13/03/17	07:02:34	ctdyndns	csq	26,99
13/03/17	07:02:34	ctdyndns	creg	0,1
13/03/17	07:02:34	ctdyndns	cops	0,0,vodafone
13/03/17	07:02:33	ctdyndns	ddns	idle
13/03/17	06:57:07	ctdyndns	csq	26,99
13/03/17	06:57:07	ctdyndns	creg	0,1
13/03/17	06:57:07	ctdyndns	cops	0,0,vodafone
13/03/17	06:57:07	ctdyndns	ddns	idle
13/03/17	06:52:04	ctdyndns	csq	26,99
13/03/17	06:52:03	ctdyndns	creg	0,1
13/03/17	06:52:03	ctdyndns	cops	0,0,vodafone
13/03/17	06:52:03	ctdyndns	ddns	idle
13/03/17	06:46:40	ctdyndns	link	dmsc:256,256,256,256,256
13/03/17	06:46:40	ctdyndns	ip	dmsc:2.45.78.231:20170313074621
13/03/17	06:46:00	ctdyndns	signal	dmsc:-84_dBm.:20170313074554
13/03/17	06:46:00	ctdyndns	csq	26,99
13/03/17	06:46:00	ctdyndns	creg	0,1
13/03/17	06:46:00	ctdyndns	cops	0,0,vodafone

Figura 2. Esempi di messaggi scambiati con il server.

4.1 Utilizzo dei messaggi per veicolare le informazioni

Per veicolare le informazioni tra il *server* centrale e i diversi *clients* distribuiti si sono scelti dei semplici messaggi come illustrato nella figura 1 e nella tabella 1. Tali messaggi vengono utilizzati dai *clients* per trasferire le informazioni diagnostiche al server che le immagazzina all'interno di una tabella. I *clients* posseggono un unico tipo di canale, quello mostrato in figura 1, mentre il server, un applicativo *FastCGI*, possiede un'ulteriore canale per trasferire le informazioni: il *web*. Se a contattarlo è uno dei *clients* software che richiede informazioni diagnostiche e decisioni amministrative, il *server*, utilizza il classico formato (figura 1). Le informazioni rilasciate sono molto compatte e un software può processarle facilmente. Se l'interlocutore è una persona fisica le informazioni vengono rilasciate, attraverso il secondo canale, nel miglior modo possibile, sotto forma di tabelle, grafici o semplici indicatori.

Il primo formato, destinato alla comunicazione esclusiva tra i moduli software, è molto simile al *CSV* [Wikipedia, Comma-separated values]. Molto più flessibile del *CSV*, in quanto possiede diversi livelli di dettaglio ed è la rappresentazione lineare di una struttura dati ad albero. Tale formato è stato adottato poiché un messaggio si articola generalmente su diversi livelli come accade in un albero. Le foglie rappresentano le informazioni mentre i caratteri speciali (i due punti, virgola, etc) rappresentano i rami, e permetteno di separare ciascun livello. Il livello più esterno utilizza le virgole come separatore, quello interno i due punti e così via. Questo formato permette di processare il messaggio molto velocemente (complessità

computazionale di tipo lineare) e di costruire istantaneamente l'eventuale albero associato. Un messaggio molto semplice riconducibile a una lista d'informazione avrà un unico livello e il formato sarà a tutti gli effetti il *CSV*. La maggior parte dei messaggi scambiati sarà di questo tipo. Tale formato si adatta bene a essere processato in automatico da qualsiasi software, anche semplice. Alcuni messaggi, come quello di *reset* sono addirittura formati da un unico parametro di uscita che può assumere solamente il valore booleano *vero* o *falso*.

Il secondo canale di comunicazione, il web, destinato alla comunicazione con le persone è molto *user friedly* poiché fa uso di elementi multimediali per trasferire le informazioni. Sono utilizzati colori per rappresentare gli stati ed eventualmente animazioni (per esempio frasi lampeggianti) per attirare l'attenzione e far risaltare informazioni. Il tutto viene ottenuto grazie alla semplicità e ricchezza delle pagine *web*. Il linguaggio H*TML* [Wikipedia, HTML] insieme ai fogli di stile a cascata (*CSS)* [Wikipedia, CSS] sono largamente utilizzati per rendere quanto più *user friendly* ed efficace l'interfaccia utente. Per completare il tutto viene utilizzato codice *javascript* che guida le scelte dell'utente ed evita che errori umani possano inficiare il funzionamento dei sistemi monitorati. Sono state utilizzate librerie molto diffuse e collaudate per raggiungere il maggior numero di utenti possibile.

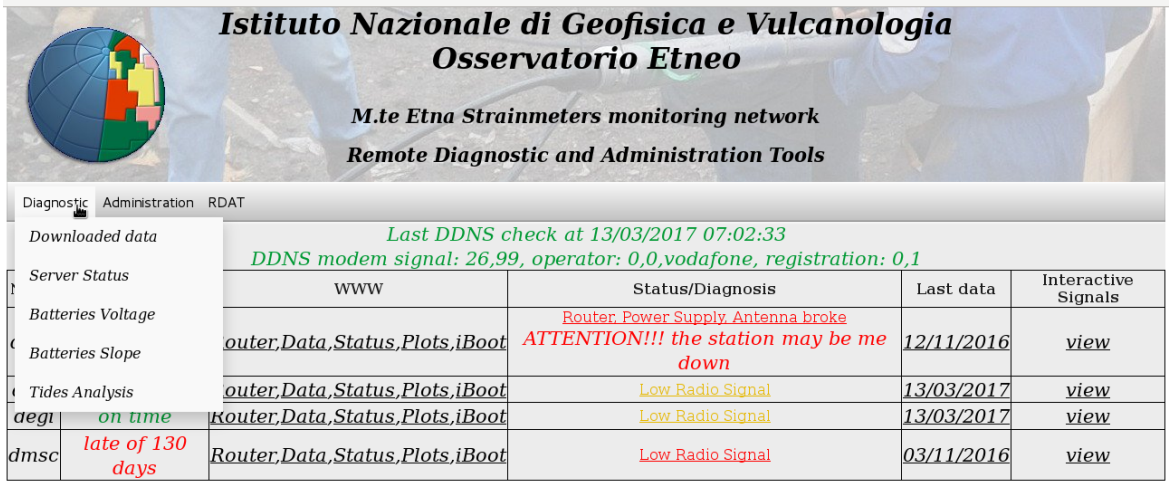

Figura 3. Pagina iniziale del Remote Diagnostic and Administration Tools (RDAT).

4.2 Sicurezza e filosofia del modello adottato

La tecnologia utilizzata per l'applicativo di tipo *server side* è il *FastCGI* [Wikipedia, FastCGI]. Il *FastCGI* è un'evoluzione del più classico *CGI* [Wikipedia, Common Gateway Interface] utilizzato da tempo per estendere le funzionalità dei server web e inserire una certa dinamicità nella fruizione delle pagine *web*. Il *FastCGI* rispetto al *CGI* classico permette di aumentare la velocità di esecuzione poiché l'applicazione rimane residente in memoria come se fosse un'estensione effettiva del server *web*. Sia il *CGI* che il *FastCGI* permettono di evitare problemi di sicurezza per gli applicativi poiché le informazioni alle applicazioni arrivano sotto forma di stringhe sullo *standard di input*. L'uscita dell'elaborazione viene restituita sullo *standard di output*. Le richieste prima di arrivare all'applicazione vengono filtrate dal server web e successivamente dalle funzioni di libreria delle *FastCGI*. All'applicazione arriverà una stringa ben formata che rappresenta i dati. L'effetto sarà quello di evitare che il software scritto in C/C++ possa essere esposto all'ambiente esterno e verrà interessato solo da problemi di sicurezza secondari. Tutto ciò a patto che i dati d'ingresso vengano processati bene e il software non abbia problemi di *stack overflow* [Wikipedia, Stack Buffer Overflow]. Tali problemi, i più difficili da trattare, nel nostro caso sono limitati al solo *server web*, alla libreria *FastCGI* e alla porzione dell'applicazione che processa i dati in ingresso (interfaccia). Una volta completata la porzione di codice addetta al *processing* dei dati in ingresso e collaudata, la manutenzione del sistema non sarà più critica e verrà delegata alla comunità internazionale che porta avanti lo sviluppo del *server web* e della libreria delle *FastCGI*. La manutenzione di un qualsiasi software, esposto al pubblico utilizzo e che abbia problemi di sicurezza, è molto costosa. Conviene nel tempo, se è possibile scegliere di relegare le modifiche alla porzione interna del software e non all'interfaccia, preservandola da errori che potrebbero sfociare in problemi di sicurezza.

L’appicazione genera delle pagine come quella mostrata nella figura 3, pensate per dare quante più informazioni sullo stato di funzionamento di una rete di monitoraggio e allo stesso tempo rimanere semplici e dirette, come il quadro di comando di un aereo. Le informazioni della pagina iniziale (figura 3) sono dati rielaborati, riscritti nel linguaggio naturale umano e danno l’idea dello stato in cui si trova la rete di monitoraggio. Dalla pagina iniziale si può eventualmente accedere alle informazioni grezze, più dettagliate che aiutano a comprendere maggiormente il perché di alcune affermazioni riportate (figura 4).

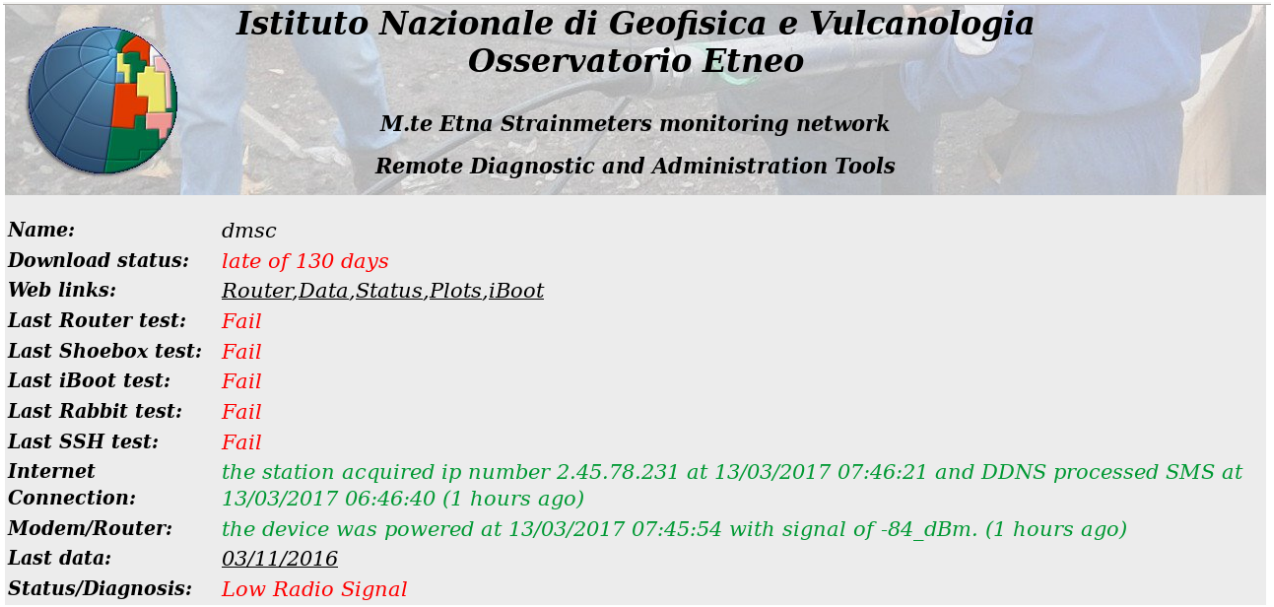

Figura 4. Dettaglio delle informazioni che hanno portato all’elaborazione dello stato di salute (health) della stazione remota.

4.3 Rapidità di accesso alle informazioni

Normalmente, quando si usano sistemi manuali, si processano tutti i dati a posteriori impiegando del tempo prezioso. La filosofia che anima l’RDAT (*Remote Diagnostic and Administration Tools*) e i sistemi automatici in genere è quella di eseguire il *processing* dei dati nel momento in cui sono disponibili. Ciò permette che l’utente trovi già tutto quello di cui ha bisogno. Oggigiorno con le risorse di calcolo mediamente disponibili e utilizzando i migliori algoritmi (stato dell’arte) possono essere eseguite analisi in tempo reale. L’accesso alle elaborazioni diviene rapido e condiviso, nel senso che anche coloro che non hanno dimestichezza con una particolare tipologia di analisi possono usufruirne, rendendosi indipendenti. Ciò permette a uno strumento di analisi di essere utilizzato in qualsiasi ora del giorno e della notte e lo rende perfetto per un utilizzo nelle Sale Operative. La rappresentazione dei dati analizzati può essere la più varia e articolata possibile, come è illustrato dalle diverse figure riportate. Per ciascun parametro si sceglierà una rappresentazione chiara ed efficace che si adatti allo scopo e al tipo di dato trattato. Si può utilizzare del semplice testo (figura 4), grafici temporali (figura 5) e perfino una rappresentazione di correlazione (figura 11). A ciascuna rappresentazione possono essere affiancati *trigger* che generano allarmi o ulteriori informazioni, che possono essere semplicemente visualizzate o inviate per essere condivise istantaneamente con tutti gli interessati.

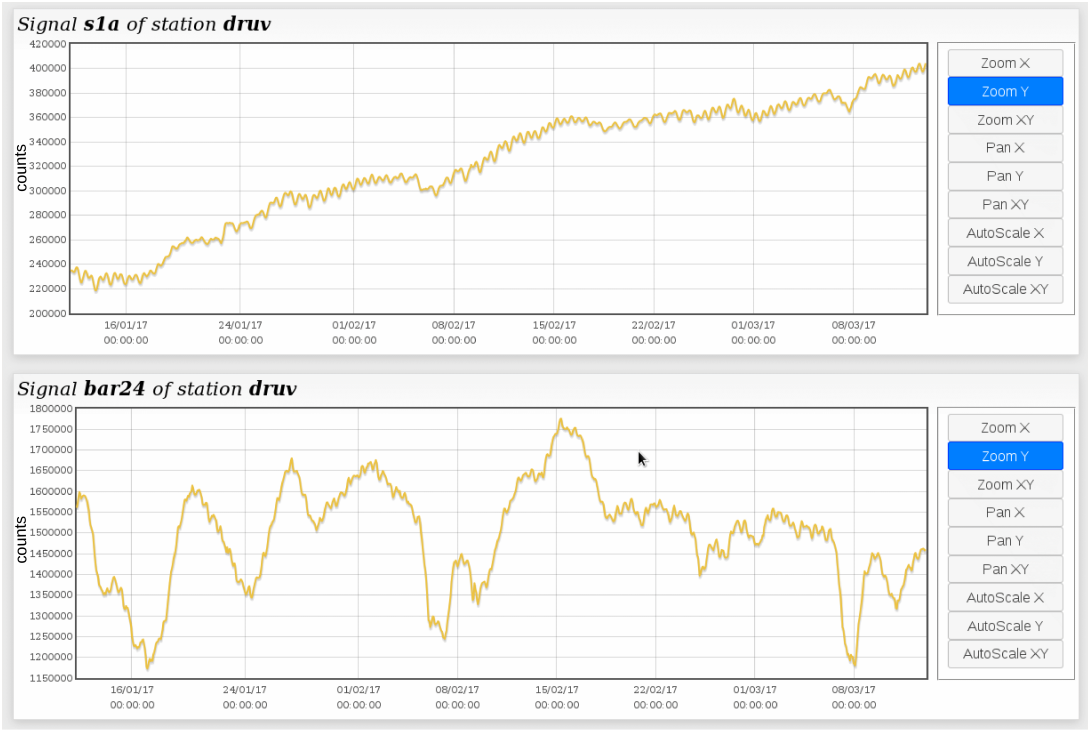

Figura 5. Rappresentazione grafica dei segnali acquisiti, organizzati per sito. Il segnale rappresenta gli ultimi 60 giorni di *strain* volumetrico e della pressione atmosferica acquisiti dalla stazione etnea DRUV.

Figura 6. Rappresentazione grafica dei segnali acquisiti, organizzati per tipologia, al fine di facilitare il confronto dello stesso segnale in siti diversi. Il segnale rappresenta gli ultimi 60 giorni della tensione di batteria acquisita nei due siti etnei DRUV (M.te Ruvolo) e DEGI (M.te Egitto).

4.4 Riconoscimento dei pattern sui messaggi

I messaggi al loro interno sono organizzati gerarchicamente per contenere informazioni. Al primo livello esistono la sorgente (*source*) che genera l'informazione, il nome del parametro (*who*) e lo stato (*what*) assunto (figura 1). All'interno di ciascun campo possono esistere altri livelli e le informazioni articolarsi nel modo più conveniente alla diagnostica applicata. Tali informazioni vengono inserite in più strutture dati e ogni qualvolta arriva un nuovo messaggio viene aggiornato lo stato di salute del sistema (*health*). Tale stato di salute comprende tutte le informazioni ricevute, rappresentate più o meno elegantemente. Ovviamente la funzionalità della rappresentazione va di pari passo con l'eleganza. Una rappresentazione che sia elegante ma non funzionale è completamente inutile. Bisogna ricercare per ogni singola informazione o gruppo, la rappresentazione più funzionale ed elegante possibile. La rappresentazione ovviamente può essere dei dati grezzi o rielaborati. Fa parte dell'insieme dei dati rielaborati la diagnosi automatica. La diagnosi viene generata attraverso il riconoscimento di *pattern* [Wikipedia, Pattern Recognition] ovvero particolari raggruppamenti degli stati assunti da tutte le informazioni diagnostiche (insieme S). In termini matematici un pattern sarebbe un elemento del prodotto cartesiano S^n. A ogni elemento del prodotto cartesiano S^n può essere associata una diagnosi. Nella realtà considerare l'insieme S^n è impossibile perchè la cardinalità [Wikipedia, Cardinality] è di tipo esponenziale poiché il numero d'informazione diagnostiche considerate (parametro n) per definizione deve essere grande. In alternativa si usa una tecnologia molto simile ai sistemi esperti [Wikipedia, Expert System] in cui la diagnosi viene ricavata attraversa una serie di domande sullo stato delle singole informazioni diagnostiche. Attraverso un numero finito di domande si riesce a far convergere la diagnosi su uno o più stati di salute del sistema. In base allo stato delle informazioni diagnostiche l'utente è informato che il sito remoto ha un particolare problema, conclamato o no, ed è necessario intervenire. Nella figura 7 è riportato un esempio di come funziona la diagnosi automatica. Lo stato di salute evidenzia che due stazioni non riescono a trasferire i dati al centro di raccolta. In mancanza di approfondimenti si sarebbero effettuati due interventi tecnici andando prima alla stazione DMSC in ritardo da più giorni. La diagnosi automatica evidenzia una situazione totalmente diversa e indica DMSC come non critica e addirittura funzionante, con problemi solamente di trasmissione. Gli interventi da due passano a uno solamente e la stazione interessata è quella che ne ha più bisogno cioè DPDN. Il sistema ha permesso di ottiene il massimo beneficio risparmiando eventuali risorse. Ciò è realizzato grazie al riconoscimento dei pattern che riesce a discriminare il comportamento delle due stazioni, associando uno stato di salute preciso nonostante all'apparenza la situazione compare completamente diversa (figura 7). L'apparenza viene completamente confutata e ridimensionate le criticità. Questo è stato possibile incrociando dati derivati da molti sistemi diversi, e riuscendo a stabilire che l'unica stazione veramente non funzionante è la prima, mentre le altre hanno solo problemi di trasmissione.

Figura 7. Menu di accesso agli strumenti di diagnostica.

4.5 Attuatori per l'amministrazione remota dei sistemi

La diagnostica può essere molto rifinita e automatizzata, ma non avrebbe alcun significato se non fosse accompagnata dalla possibilità d'intervenire per risolvere i malfunzionamenti che man mano vanno generandosi. Per questo ogni strumento di diagnostica deve essere affiancato in simbiosi perfetta da un sistema di amministrazione. Chi ha studiato controllo di processi sa benissimo che i sistemi stabilizzati con l'utilizzo di tecniche di retroazione negativa [Wikipedia, PID Controller] posseggono due parti fondamentali: una di misura e l'altra di attuazione. Quando un qualsiasi sistema esce dal suo funzionamento normale e l'anomalia viene riscontrata attraverso delle misure (diagnostica) è compito dell'attuatore (amministrazione) riportarlo alla normalità. Gli attuatori a disposizione dello strumento di amministrazione possono essere azionati in automatico o in manuale. Gli attuatori automatici possono essere attivati preventivamente o su particolari segnali. Gli attuatori manuali possono essere attivati dall'amministratore del sistema o possibilmente anche da utenti comuni coadiuvati da uno o più segnali diagnostici. Fanno parte degli attuatori automatici preventivi i reset per i sistemi di trasmissione [Sicali et al., 2016]. I *watchdog* che si utilizzano normalmente nei sistemi di acquisizione fanno parte degli attuatori automatici coadiuvati dalla diagnostica. Per esempio nel sistema *Mag-Net* [Del Negro et al., 2002] l'acquisizione è sorvegliata da un *watchdog* hardware [Sicali et al., 2016] che interviene se il normale funzionamento viene interrotto. Gli attuatori automatici sono da preferire a quelli manuali poiché hanno tempi d'intervento molto ridotti. Si possono usare procedure manuali o semi-manuali per i casi in cui non è possibile automatizzare completamente l'amministrazione dei sistemi. L'efficacia dei sistemi amministrativi manuali è tanto più alta quanto più facilmente sono accessibili. Ovviamente sono da preferire sistemi quasi completamente automatizzati che richiedono solamente un segnale di attivazione da parte dell'utente. Una complessa procedura di diagnostica è inefficace se poi per metterla in atto ci vogliono risorse non disponibili. L'intervento diviene più semplice se tale procedura può essere completamente (o quasi) automatizzata. Del resto per definizione ogni procedura è formata da una successione di passi ed è molto probabile che possa essere codificata già in un algoritmo. L'utente ha solo l'onere di agire su un pulsante per attivarla. L'efficienza dei sistemi di amministrazione raggiunge il top se per attivarli si utilizzano canali di comunicazione comuni e globali come il web, come le applicazioni mobili (*app*) o le e-mail. L'amministratore resosi conto di un particolare malfunzionamento può interagire con il sistema di amministrazione attraverso l'app o l'email di notifica contenente codice *HTML* e *javascript*. Il tutto si ridurrà semplicemente a pigiare un tasto e il problema verrà risolto. Ovviamente il punto cruciale sarà accorgersi di un malfunzionamento. Per questo è meglio che i sistemi siano di tipo asincroni e capaci di notificare il problema. Il sistema realizzato (*RDAT*) è multicanale e prevede entrambi i tipi di funzionamento (asincrono e sincrono). Dispone di un sito web, della possibilità di spedire notifiche attraverso e-mail e si avvale anche di un'app.

4.6 Funzionamento asincrono grazie all'invio delle email

I sistemi sincroni sono da sempre grandi consumatori di risorse e non assicurano il buon funzionamento nel tempo. Può accadere che il sistema abbia un malfunzionamento tra un controllo e l'altro. Se i controlli sono molto diradati nel tempo si scoprirà tardi di un particolare malfunzionamento. I sistemi asincroni e i sistemi orientati agli eventi [Wikipedia, Event-driven programming] stanno sostituendo quasi completamente i sistemi sincroni. Mentre il sito web è un strumento di tipo sincrono e dovrà essere l'utente a consultarlo per verificare un eventuale malfunzionamento, le e-mail sono asincrone: hanno la capacità di avvertire l'utente nel momento in cui accade il problema. Ovviamente questo metodo ha senso oggi, vista la possibilità di lettura di email dal proprio *smartphone*; in tempi passati sarebbero stati utilizzati altri canali informativi come ad esempio gli *SMS* (Short Message Service). L'utilizzo delle e-mail permette di ottenere uno strumento più flessibile rispetto agli *SMS* poiché è possibile spedire qualsiasi tipo d'informazione, anche audio-video, basso costo o addirittura nullo. Le e-mail possono contenere informazioni di qualsiasi tipologia che possono chiarire la situazione. Possono essere spedite immagini che rappresentano l'anomalia rilevata o semplicemente lo stato di salute delle stazioni (*health*). Le e-mail possono essere personalizzate in base al tipo di utente. Le informazioni riportate all'amministratore saranno più articolate e complesse, rispetto a quelle inoltrate a un utente comune, per aiutare a comprendere il problema e formulare una prima idea d'intervento. Possono essere spedite solo nei momenti critici o anche periodicamente per informare che tutto proceda per il meglio. Nel corpo dell'email potrebbero essere inserire anche domande che il sistema pone e a cui non riesce a dare una risposta sicura. I sistemi semi-automatici in alcune situazioni potrebbero arrivare a un bivio decisionale e non riuscire a prendere una decisione autonomamente. Potrebbe essere compito solo dell'amministratore decidere se resettare o meno un server o un sistema remoto di fronte ad un quadro clinico impreciso. Il sistema di amministrazione potrebbe spedire tutte le informazioni via e-mail in formato

HTML, ponendo una semplice domanda e la possibilità per mezzo di semplici tasti di scegliere la strada da seguire. Questi tasti gestiti attraverso il linguaggio di programmazione *JAVASCRIPT* forniscono la risposta al sistema di amministrazione inoltrando un semplice messaggio (figura 1) al server *FastCGI*. Tale messaggio verrà usato per proseguire nella procedura semi automatica avviata precedentemente.

5. Analisi dei segnali per estrarre informazioni utili alla diagnostica

L'analisi dei segnali può essere utile per diagnosticare problemi ai sistemi di monitoraggio. Ad esempio l'analisi di trend sul segnale o del contenuto spettrale può evidenziare problemi ai convertitori analogico digitali (*ADC*). Analisi della pendenza della scarica nottura (*slope*) della tensione di alimentazione può evidenziare problemi al sistema energetico, non solo in atto ma anche futuri, come pannelli solari guasti o batterie esauste. L'analisi della qualità del segnale radio può evidenziare problemi al sistema di trasmissione come un connettore spezzato o un'antenna ossidata e/o danneggiata.

5.1 Analisi della scarica notturna delle batterie

La manutenzione della parte energetica di un sistema di acquisizione remoto ricopre un ruolo di fondamentale importanza. Generalmente la strumentazione elettronica può avere una vita operativa anche di decine di anni. Purtroppo non si può dire la stessa cosa delle batterie che risentono molto dell'usura del tempo. I problemi che affliggono le batterie sono molteplici e la loro vita media spesso non supera i due anni. Un problema che affligge ogni tipologia di batterie è la riduzione della capacità nominale. In alcuni situazioni la disponibilità di carica può bloccare per periodi più o meno lunghi il normale funzionamento di un sito remoto di acquisizione, a causa soprattutto del maltempo se i sistemi sono alimentati a energia solare. Se lo stadio di alimentazione è stato progettato per resistere a un certo numero di giorni di maltempo, quando la capacità nominale delle batterie diviene troppo bassa si può spegnere tutto il sistema. Per evitare blackout energetici in momenti inopportuni, ovvero quando il sito potrebbe essere irraggiungibile, è necessario prevedere prima tali malfunzionamenti. In figura 8 sono riportate le tensioni di batteria di due stazioni magnetiche dislocate sul M.te Etna a una quota otre i 2500 s.l.m. e una (segnale verde) sui monti Nebrodi a 1300 s.l.m. Le tre stazioni durante l'inverno sono ovviamente irraggiungibili a causa della neve.

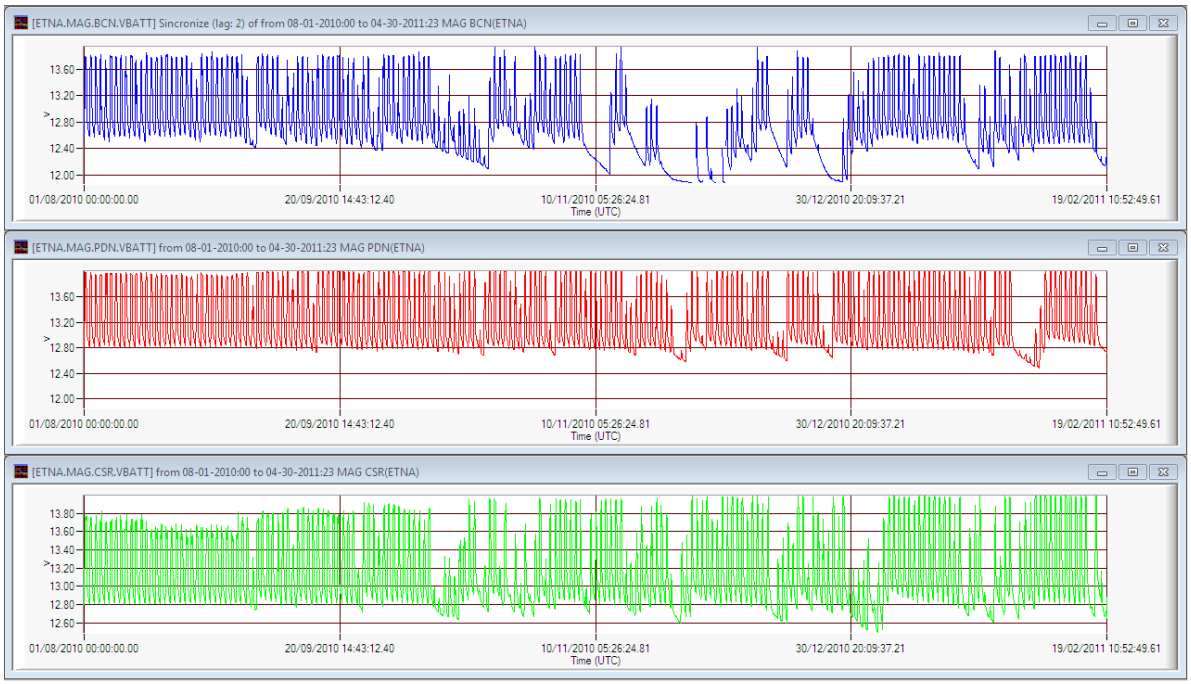

Figura 8. Tensione di batteria misurata in tre stazioni a cavallo della fine dell'anno solare.

I tre segnali coprono un periodo molto critico per le stazioni di monitoraggio ovvero la conclusione dell'anno solare. Il maggior numero di blackout energetici avviene in questo periodo a causa delle cattive condizioni meteorologiche e per la presenza di neve. I sistemi possono spegnersi anche in condizioni normali se il sistema energetico non è efficiente. La stazione BCN (segnale blu) verso la fine dell'anno ha avuto problemi di alimentazione mentre le altre due hanno funzionato normalmente, con qualche crisi energetica ma senza spegnersi completamente. Apparentemente nella prima parte del grafico i tre segnali sembrano uguali, sebbene la tensione media è più bassa per la stazione BCN (segnale blu). Generalmente la tensione, legata attraverso una relazione non lineare alla carica residua, dipende dalla storia recente di maltempo che ha interessato il sito e non può essere usata per stabilire lo stato di salute delle batterie. Effettuando uno *zoom* della prima parte dei segnali si ottiene la figura 9. In tale figura si può notare che qualcosa di nettamente diverso esiste tra i tre segnali: la stazione BCN ha delle scariche notturne molto più intense. La variazione di tensione dipende dalla corrente erogata e dalla carica nominale della batteria, caratteristiche simile nei tre casi. La stazione BCN, nonostante abbia lo stesso sistema energetico delle altre, non si comporta nel medesimo modo, ovvero non mostra la stessa efficienza. Inizialmente si è realizzato uno strumento che monitora l'efficienza di batteria attraverso una misura della pendenza. La pendenza della scarica notturna (*slope*) viene calcolata ogni qualvolta vengono resi disponibili dati e creato un grafico che mostra l'andamento nel tempo del comportamento di scarica-carica delle batterie (figura 10). Lo strumento può effetivamente rilevare se esiste un'efficienza energetica nelle batterie attraverso soprattutto il confronto dei segnali con sistemi energetici simili (figura 10). Dalla figura 10 per confronto, considerando che le due stazioni hanno un sistema energetico identico, si può stabilire che le batterie di DRUV (grafico in alto) sono meno efficienti di DEGI. Tutto ciò viene confermato dall'età delle batterie maggiore per la stazione DRUV. Tuttavia dai grafici è molto difficile stabilire oggettivamente l'opportunità di sostituire le batterie o meno poiché non esistono soglie prestabilite e uno studio approfondito del fenomeno. Per questo si è corredato lo strumento di un'altra analisi più statistica che si basa sulla correlazione tra il minimo e massimo giornaliero.

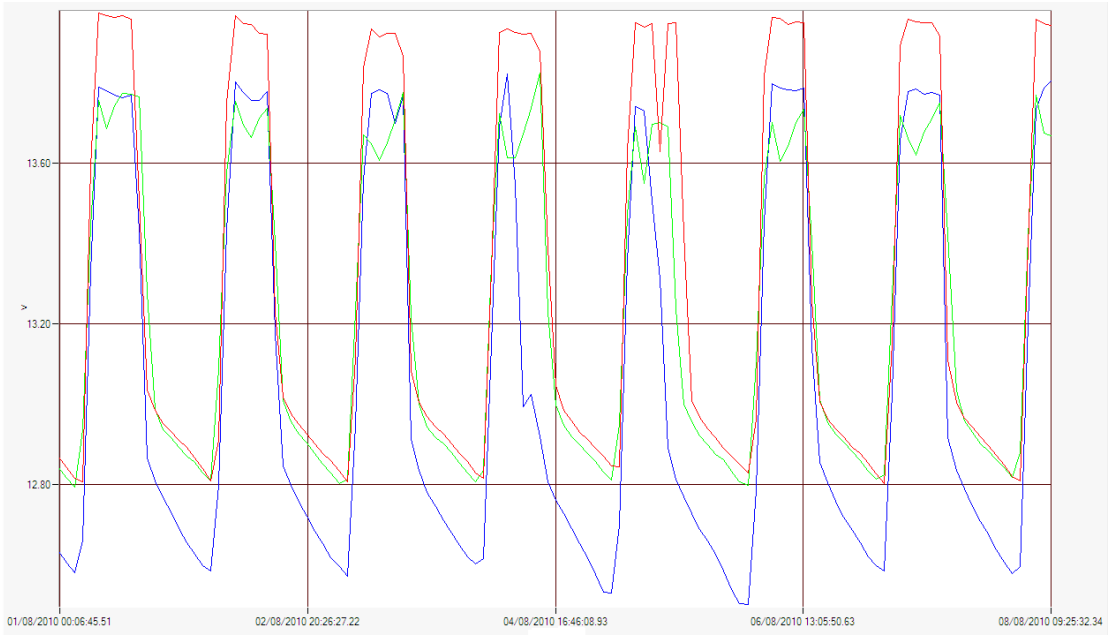

Figura 9. Dettaglio della scarica notturna.

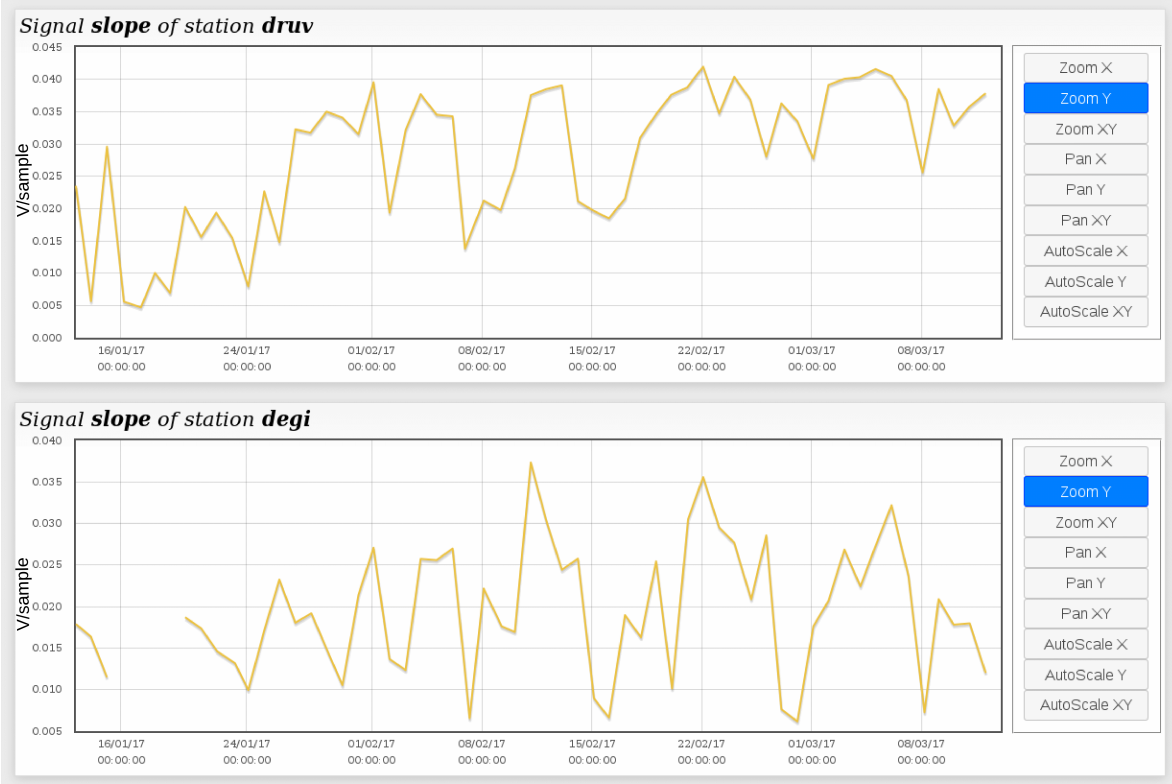

Figura 10. Rappresentazione grafica della pendenza (*slope*) della scarica notturna delle batterie. Notare la possibilità che le batterie della prima stazione (DRUV) possano essere meno efficienti rispetto alla seconda (DEGI).

5.2 Analisi di correlazione minimo/massimo giornaliero

L'analisi della pendenza della scarica notturna è sicuramente uno strumento promettente per rilevare problemi alle batterie, l'unico inconveniente e che è difficile prendere decisioni operative a causa della non semplice interpretazione della curva. Ci permette di stabilire se le batterie sono inefficienti ma non di stabilire quando sostituirle, a causa della mancanza di soglie prestabilite. Più fruttuoso è utilizzare un altro strumento, con carattere più statistico e di semplice interpretazione.

Lo strumento si basa su una semplice osservazione compiuta sulla tensione di batteria subito dopo una ricarica. Dopo una ricarica le batterie al piombo mostrano generalmente una riduzione di tensione fisiologica, anche se non è connesso alcun dispositivo (autoscarica). In alcune batterie, quelle danneggiate, la tensione si riduce così tanto da riportarsi ai valori originari. La capacità di immagazzinare l'energia su lungo periodo viene quasi completamente eliminata e la batteria si comporta come un grosso condensatore che purtroppo non mantiene la carica nonostante la tensione abbia raggiunto il suo massimo. La poca carica immagazzinata viene successivamente persa nei minuti e nelle ore successive per autoscarica. Lo stesso effetto che viene visto durante l'analisi della scarica nottura come incremento dello *slope*. Tale comportamento è osservabile anche quando ricarichiamo il nostro cellulare e dopo poche ore è già alla massima tensione di batteria, tanto che il dispositivo ci avverte che la carica è finita. L'utilizzo di qualche ora, riporta la tensione al minimo e viene indicata la batteria come scarica. In un periodo molto breve si è avuto una forte oscillazione di tensione. Sono queste oscillazioni che vengono osservate attraverso lo strumento proposto. Lo strumento presentato semplicemente calcola il massimo e il minimo giornaliero della tensione di batteria e lo riporta in un grafico di correlazione (figura 11). Come si vede dalla figura 11 i punti possono cadere in aree la cui colorazione è bianca, rossa, gialla e verde. Le aree sono state fissate in base all'esperienza maturata sulle batterie al piombo la cui tensione nominale è di 12 volt e possono essere allargate o ristrette secondo le esigenze. Un invariante di tale tecnica è rappresentato dalla posizione relativa dei punti rispetto agli angoli. Quanto più un punto si avvicina a uno degli angoli tanto più viene confermato quel tipo di comportamento. Le aree bianche sono prive di un significato particolare, i punti ricadenti in tali aree corrispondono ad un comportamento neutro (né buono né cattivo). L'area verde indica una batteria con minimi e massimi alti, in ottimo stato di funzionamento e in piena carica. I punti ricadenti nell'area gialla

indicano una batteria con minimi e massimi bassi, buona ma scarica. L'area più interessante che prelude un comportamento capacitivo e una riduzione della carica nominale è quella rossa (massimi alti e minimi bassi). Quanti più punti si addensano nella stessa area tanto più la batteria assume quel tipo di funzionamento. Si noti che non è necessario sapere se i punti dell'area rossa sono precedenti a quelli delle altre aree, il solo fatto che esistono dovrebbe far pensare ad un malfunzionamento della batteria. Nel grafico della figura 11 ci sono due punti che ricadono nell'area rossa, uno sul confine e almeno quattro in prossimità si può iniziare a pensare di sostituire le batterie in tale stazione. Infatti tale grafico è stato realizzato per il segnale blu della figura 8 (stazione BCN). Il grafico di correlazione per uno degli altri segnali è riportato nella figura 12. Si può notare come sia netta la differenza.

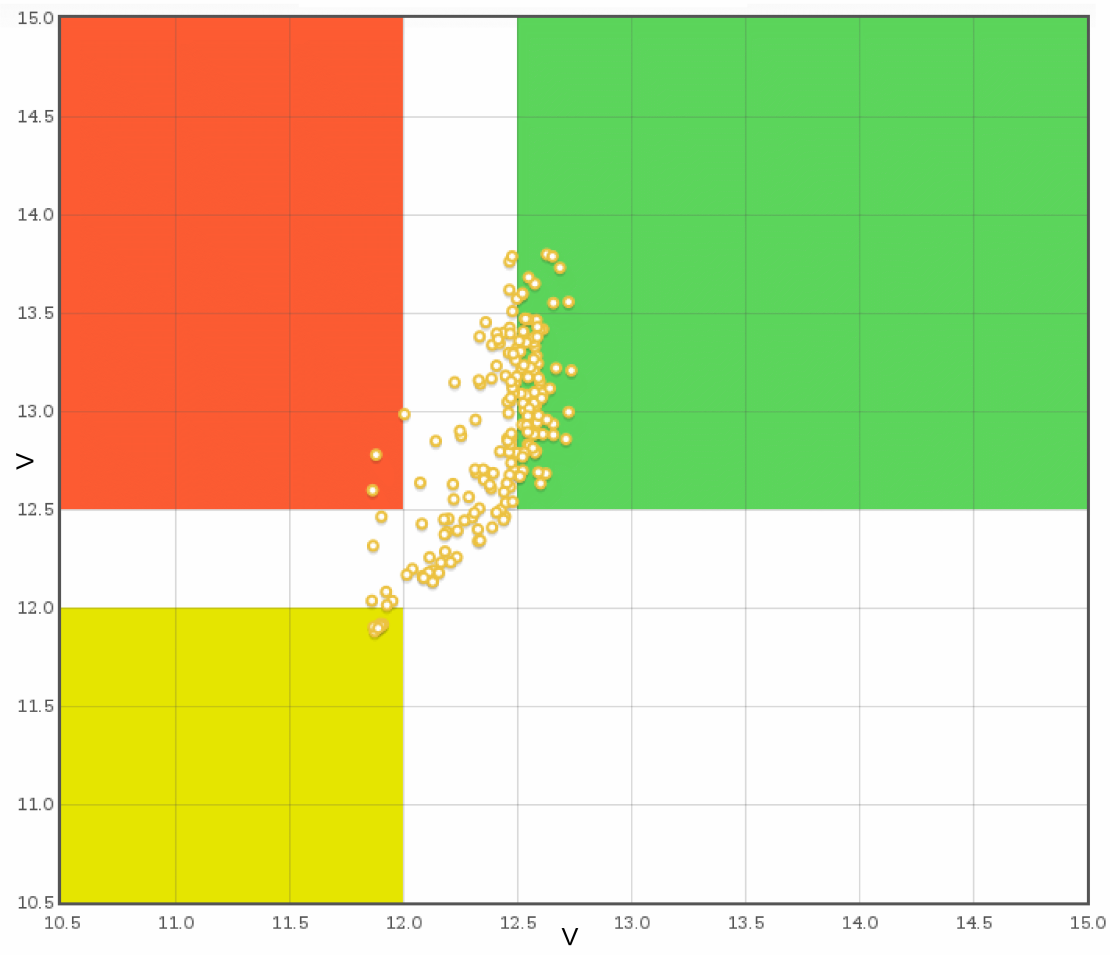

Figura 11. Analisi di correlazione minimo/massimo giornaliero per il sistema di alimentazione della stazione BCN (batteria in cattivo stato).

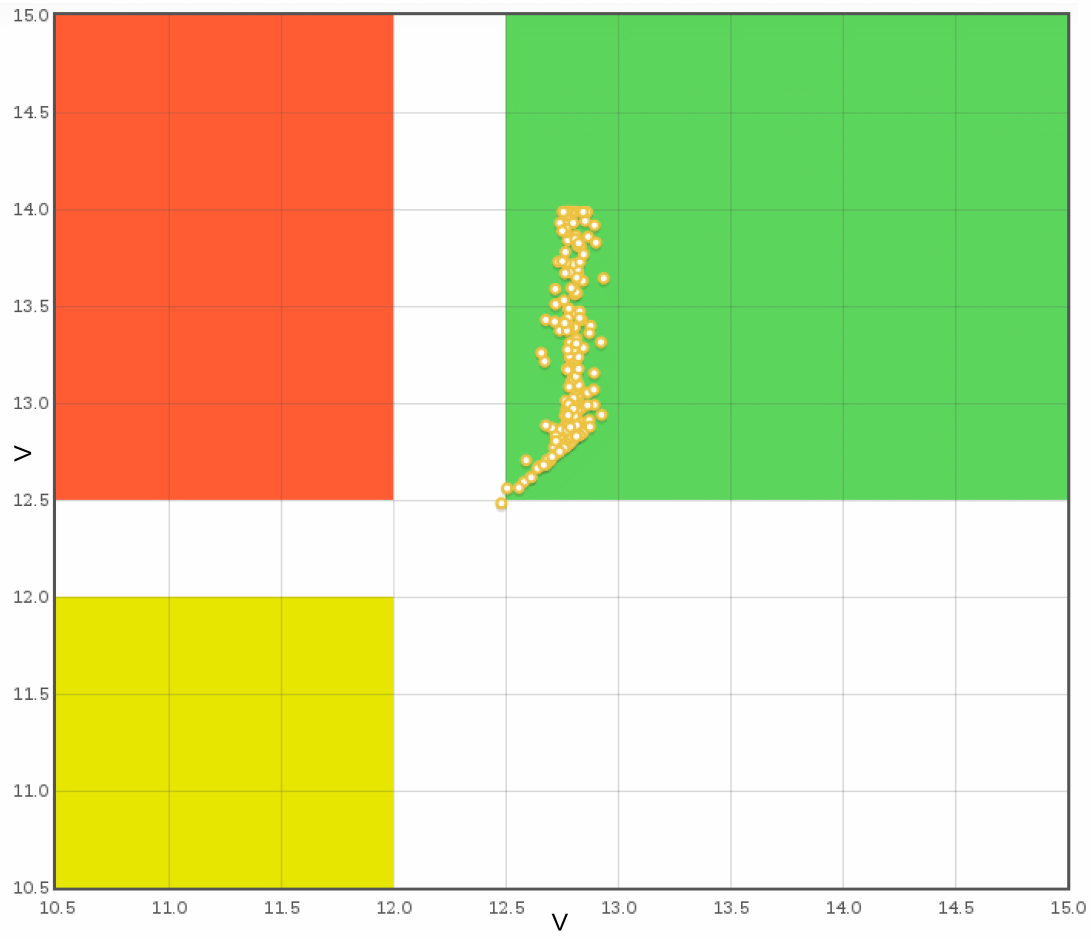

Figura 12. Analisi di correlazione minimo/massimo giornaliero (batteria in ottimo stato).

5.3 Analisi del segnale principale (analisi delle maree)

Quando si posseggono un numero elevato di stazioni di acquisizione è importante capire se queste stiano misurando correttamente e se il segnale sia di qualità accettabile. L'operazione di verifica oltre ad essere *time consuming* può essere anche difficoltosa da eseguire per alcuni segnali indistinguibili dal rumore di fondo. Le idee proposte per il segnale studio, lo *strain* volumetrico, possono essere estese ad altri tipi di segnali.

Molte volte di fronte a un cambiamento brusco del segnale acquisito ci si trova in serie difficoltà per stabilire se sia successo qualcosa alla strumentazione. Non si riesce a capire se il nuovo segnale sia completamente genuino. Andare a stabilire se un segnale è genuino, senza strumenti oggettivi, è praticamente impossibile e soprattutto il tempo per le elaborazioni rende la verifica molto difficile per un numero elevato di siti. Il segnale generato da un dilatometro [Sicali et al., 2013] possiede tra le sue componenti una periodica che rappresenta le maree terrestri. Tale segnale, che in alcune stazioni è più spiccato che in altre, è sempre presente ed è segno di buon funzionamento. Si è elaborato uno strumento che permettesse di rilevare la presenza di tale componente attraverso una semplice analisi di correlazione. Lo strumento software proposto genera un segnale oscillante (figura 13) e permette di stabilire la genuinità dell'acquisizione attraverso lo studio delle maree. Dalla figura 13 si può osservate come non tutte le stazioni contengono un buon segnale di marea. In presenza di una cattiva componente mareale l'oscillazione verrà interrotta molte volte (secondo segnale della figura 13). Nonostante ciò le oscillazioni saranno sempre presenti. In presenza di un problema più serio e di un falso segnale le oscillazioni saranno molto meno evidenti e cambieranno totalmente il loro ritmo assumendo la forma del riquadro rosso della figura 14.

Figura 13. Rappresentazione grafica dell'oscillatore rivelatore di maree.

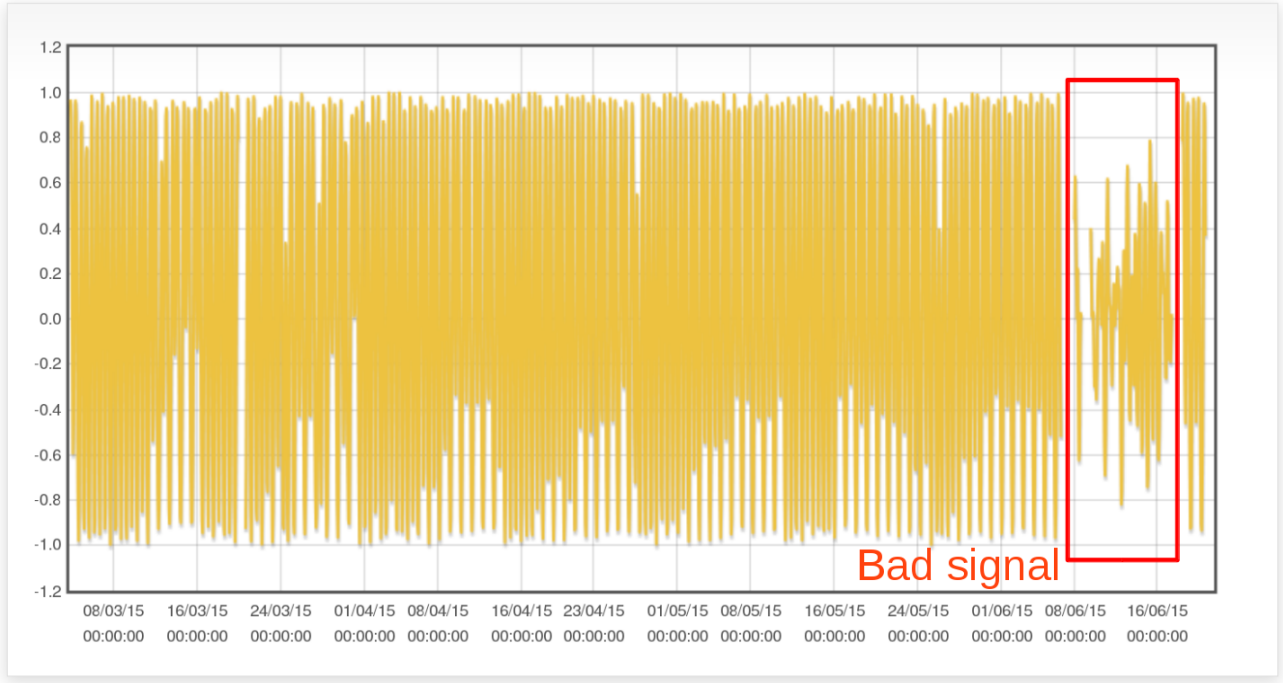

Figura 14. Rappresentazione grafica dell'oscillatore rivelatore di maree in presenza di un segnale corrotto.

6. Applicazione *Android*

L'utilizzo di un sito web per la diagnostica e l'amministrazione remota dei sistemi indubbiamente permette di ottenere globalità e centralità. L'unico inconveniente nell'usare un sito web sta nella praticità e nell'interattività rispetto a un applicativo. Storicamente le pagine *web* sono sempre state più lente e letteralmente ingessate rispetto a un applicativo che viene eseguito localmente. I siti *web* più recenti possono utilizzare tecnologie molto sofisticate per aumentare il grado di interattività [Currenti et al., 2014]. Nonostante ciò rimarranno sempre un passo indietro rispetto a una applicazione *standalone*. Inoltre un'applicazione *standalone* permette di ridurre al minimo lo scambio di dati con un server remoto anche non facendo uso di tecniche di *caching*. In alcune situazioni, durante periodi o in luoghi particolari, potrebbe essere difficile visualizzare un'intera pagina *web* rendendo l'interazione impossibile. In tali momenti potrebbe essere necessario utilizzare un applicativo *standalone* per scambiare solo pochi byte essenziali al mantenimento della diagnostica e manutenzione. Da molti anni è possibile programmare i dispositivi mobili come *tablet* e *smartphone* per premetegli di esplicare qualsiasi funzione si voglia. Si può trasformare uno *smartphone* in qualsiasi strumento si abbia la necessità, basta installare un'applicazione, semplicemente una *app* [Wikipedia, Mobile app]. Una *app* può trasformare un dispositivo in qualsiasi strumento si necessiti. Uno *smartphone* può diventare una bussola, una livella, un fonometro o navigatore *GPS*. Nel nostro caso il dispositivo viene trasformato nel quadro di controllo della rete di monitoraggio. Un quadro di controllo molto semplice che permette di ricevere informazione istantaneamente e d'intervenire altrettanto rapidamente. Come mostrato nelle figure 15 e 16 l'applicativo è molto semplice, più del sito web già di per sé semplice. Tramite questa *app* vengono riassunte le informazioni sullo stato di funzionamento nel modo più chiaro e sintetico possibile.

L'applicazione si compone di due sezioni che ricordano le due principali funzioni dell'*RDAT*: la diagnostica e la manutenzione. Si può passare da una sezione all'altra attraverso un'operazione velocissima di *swipe* orizzontale [Wikipedia, Multi touch] tipica degli schermi *touch*.

La sezione di diagnostica (figura 15) possiede una riga di stato per ciascun sito monitorato. In coda alla pagina è presente lo stato di funzionamento del sistema di *Dynamic DNS* [Sicali et at, 2017]. Ciascuna riga contiene il nome del sito (in blu), il numero di giorni di ritardo accumulato sui dati e un'icona rappresentante lo stato generale del sito. Ogni riga sfrutta l'espressività dei colori per esprimente ancor maggiormente lo stato di salute (*health*). Sono stati individuati sei livelli generali di salute, come riassunto dalla tabella 2. Solo uno dei sei è critico per il buon funzionamento delle stazioni, quello rappresentato dalla pila rossa. Ciascuna icona rappresenta un tasto su cui è possibile agire per chiedere informazioni aggiuntive sullo stato di salute, sulla diagnostica e su eventuali criticità (figura 15).

Normalmente l'*RDAT* sorveglia anche il buon funzionamento dei server e può riavviarli in automatico in presenza di malfunzionamenti. Può accadere però che in presenza di errori di programmazioni o per casistiche mai considerate i server entrino in stallo. In tale situazione può essere utilizzata la seconda sezione dell'*app*: l'amministrazione remota (figura 16). Agendo su uno dei due pulsanti, e confermando la propria volontà rispondendo alla domanda posta (figura 16), viene iniziato il riavvio del server scelto.

Grazie all'uso dei dispositivi mobili la diagnostica e l'amministrazione sono divenute ancor più semplici e immediate: con solamente due schermate di *smartphone* o *tablet* si riesce a gestire un'intera rete di monitoraggio.

Figura 15. Diagnostica remota attraverso l'applicazione *Android - RDAT*.

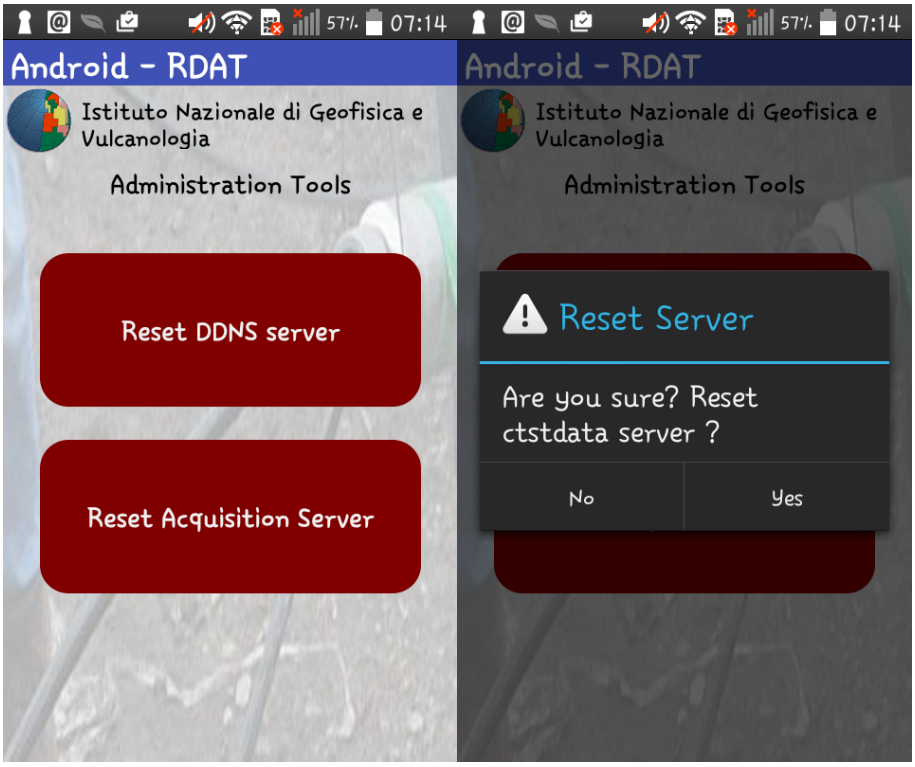

Figura 16. Amministrazione remota attraverso l'applicazione Android -RDAT.

Icona	Stato	Descrizione
	Ottimo	Il sito è in ottime condizione di salute
	Acquisizione ferma	Il sito è afflitto da un problema serio che può compromettere il normale funzionamento
	Problemi all'archiviazione	Il sito è afflitto da un problema serio che può compromettere l'archiviazione del dato
	Problemi di accesso informatico	Il sito è afflitto da un problema di accessibilità dei servizi di informatici
	Problemi di trasmissione	La trasmissione non è al massimo dell'efficienza. Difficoltà riscontrata nelle trasmissioni.
	Seri problemi di trasmissione	La trasmissione ha serie difficoltà.

Tabella 2. Stati di salute dell'Android RDAT e relative icone rappresentative.

7. Installazione e utilizzo

Il software ha una struttura distribuita e non monolitica. Esistono diversi moduli che dovranno essere installati secondo le esigenze. Il modulo principale riportato nelle appendici A1 e A2 produce le pagine web di dialogo con l'utente, elabora le diagnosi, attua le procedure di amministrazione automatica e invia eventualmente le *email*. Tale modulo rappresenta il server del modello descritto. Nelle appendici A5-A6 è stato riportato un modulo che invece genera messaggi e segnali utili al software per funzionare (sensore di diagnostica). Tale modulo è solo un esempio di ciò che è possibile trasferire attraverso i messaggi al server centrale. Nelle appendici A7 e A8 è riportato un modulo che permette di resettare i server attraverso un processo sicuro (attuatore di amministrazione). Tale modulo, l'unico di tutto il sistema che utilizza il *polling*, interroga ciclicamente il modulo centrale dell'RDAT per sapere se bisogna resettare il *server* locale. Il polling in questo caso è stato necessario per evitare problemi di sicurezza e mantenere isolati i *server* interessati, allo stesso modo in cui gli *SMS* rendono il DDNS sicuro [Sicali et al., 2017]. Sostituendo l'interrogazione in polling del *web* con i messaggi tipo *SMS* si potrebbe rendere il tutto asincrono però con la conseguenza di aumentare la complessità del sistema. La maggiore complessità renderebbe il tutto potenzialmente instabile [Sicali et al., 2016]. Conviene sicuramente mantenere il modulo in *polling*, nella realtà non consuma molte risorse poiché possiede un ciclo molto lento, circa 10 secondi.

Oggetto	Tipo	Descrizione
/images/	directory	Contiene le immagini di supporto
/images/bar.jpg	file	Sfondo della barra superiore
/flot/	directory	Contiene la libreria javascript per la visualizzazione dei grafici, reperibile all'indirizzo web http://www.flotcharts.org/
/jquery/	directory	Contiene la libreria javascript per la visualizzione dell'intero sito web, reperibile agli indirizzi https://jquery.com/ e https://jqueryui.com/
/web.admin.css	file	Foglio di style per la visualizzazione del sito web

Tabella 3. Struttura del sito web utilizzata dal server FCGI per generare le pagine dinamiche.

7.1 Installazione e descrizione modulo centrale FastCGI

Nelle appendici A1 e A2 sono riportati i codici sorgenti del server *FastCGI*. Nell'appendice A3 è riportato il *Makefile* necessario alla compilazione e all'installazione del server. Tale *file* dovrà essere eseguito con i massimi privilegi (*root)* poiché esegue operazioni riservate agli amministratori, come il riavvio del *server web*. Il *server web* utilizzato è Apache nella release 2, ma nulla vieta di utilizzarne qualsiasi altro che disponga l'interfaccia *FCGI*. Una volta compilato e installato il server non ha bisogno di null'altro per funzionare. Diversamente dal lato client le pagine *web* generate hanno bisogno di alcune librerie javascript e di alcune immagini di supporto da inserire nel *server web*. Nella tabella 3 sono riportate le *directory* di cui è composto il sito *web*. Tale struttura è ciò che il software nella versione attuale utilizza. Non è l'unica e può essere modificata a piacere di pari passo al software. Come si può notare ci sono *file* appartenenti a tre diverse librerie javascript: jquery [Wikipedia, JQuery], jquery ui [Wikipedia, JQuery UI] e flot [Flot]. Nell'appendice A4 è riportato infine un esempio di *file CSS* (Cascading Style Sheets) che è possibile modificare per raggiungere il risultato grafico desiderato.

Il software come mostrato nella figura 17 e dalla tabella 4 possiede 6 funzioni che visualizzano informazioni e una soltanto di elaborazione (*insert*). Le funzioni di visualizzazione costruiscono semplicemente delle pagine di tipo *HTML* anche se sembrano molto confuse all'apparenza.

L'unica funzione di elaborazione (*insert)* si occupa invece di elaborare le informazioni in arrivo e lo stato di salute (*health*). Nella figura 18 è riportato un diagramma di flusso della parte software derivata dalla funzione *insert* e che si occupa delle elaborazioni. Dopo l'inserimento del messaggio nel *database* viene avviata la funzione *processInsert* che elabora l'informazione ricevuta. L'informazione rielaborata viene trasferita alla funzione *Diagnosis* che produce la diagnosi automatica e invia eventualmente le email di notifica.

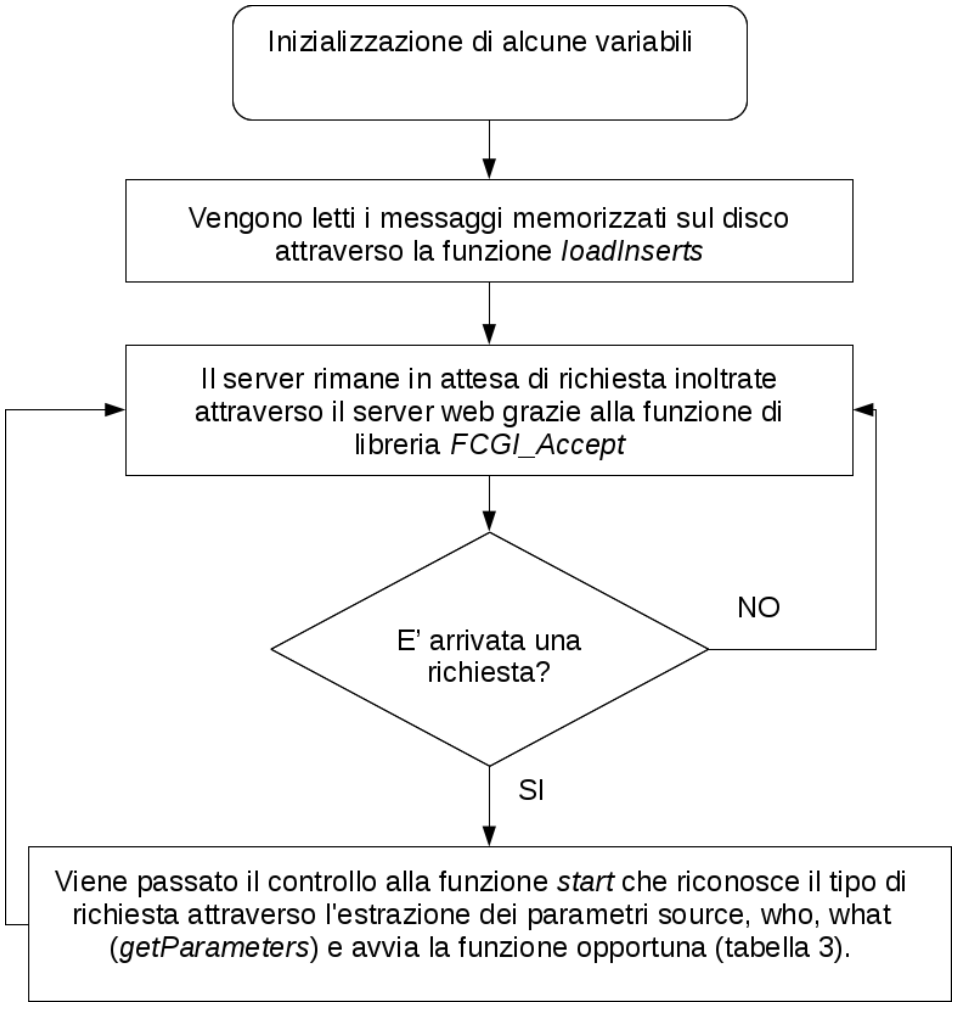

Figura 17. Diagramma di flusso del server FCGI.

Nome funzione	Descrizione
insert	Processa un nuovo messaggio arrivato
	Sfondo della barra superiore
getReset	Ritorna la presenza di richieste pendenti di reset
showSignals	Mostra i grafici dei segnali acquisiti o elaborati
showServers	Mostra informazioni sullo stato dei server
showInserts	Mostra gli ultimi messagi arrivati
showSyntesis	Mostra una sintesi dello stato di salute delle stazioni (*health*). Viene usata dall'applicazione Android RDAT
showDetails	Mostra i dettagli dello stato di salute (*health*)

Tabella 4. Lista delle funzioni componenti il modulo centrale (server *FCGI*).

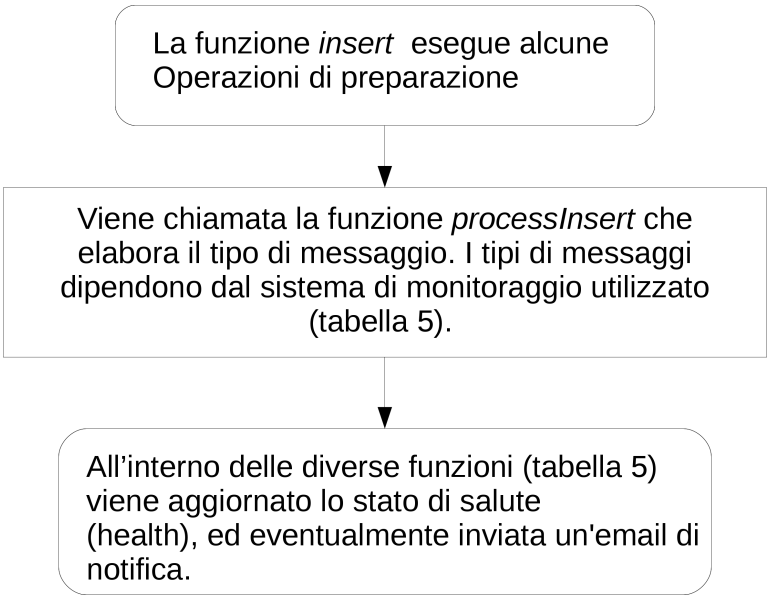

Figura 18. Diagramma di flusso elaborazione stato di salute (*health*).

Nome funzione	Utilizzo	Descrizione
setReset	Generale in tutti i sistemi	Processa una richiesta di reset diretta ad un server
setIP	Specifico per alcuni sistemi	Acquisisce l'indirizzo IP dinamico che alcuni siti potrebbero avere
setSignal	Specifico per alcuni sistemi	Acquisisce la qualità del segnale di un sito se disponibile
setDelay	Generale in tutti i sistemi	Acquisisce il ritardo nel trasferimento dei dati
setLink	Specifico per alcuni sistemi	Acquisisce informazioni sullo stato di collegamento dei sistemi
setDDNS	Specifico per alcuni sistemi	Acquisisce lo stato del sistema di DDNS [Sicali et al., 2017]
Diagnosis	Generale in tutti i sistemi	Elabora una diagnosi di intervento
sendMail	Generale in tutti i sistemi	Spedisce le e-mail di notifica

Tabella 5. Lista delle funzioni utilizzate per l'elaborazione dello stato di salute (*health*).

7.2 Installazione e descrizione clients diagnostici

Nelle appendici A5 e A6 è riportato uno dei *client* che produce informazioni diagnostiche dirette al server *FCGI*. Questo *client* nello specifico permette di misurare i giorni di ritardo accumulati dalla procedura di download. Nella figura 19 è riportato un diagramma di flusso del software. Come si può notare la struttura dei *clients* che producono segnali di diagnostica è molto semplice e si riduce ad un'unica operazione. Tutti i *clients* diagnostici interrogano semplicemente il server *FCGI* attraverso una connessione con protocollo HTTP e una richiesta di tipo *GET* avente sempre la struttura riportata nella figura 19. Tale richiesta nonostante all'apparenza sia una lettura, permette di comunicare al modulo centrale una particolare informazione. Il resto delle operazioni compiute dal software sono proprie del singolo modulo di diagnostica.

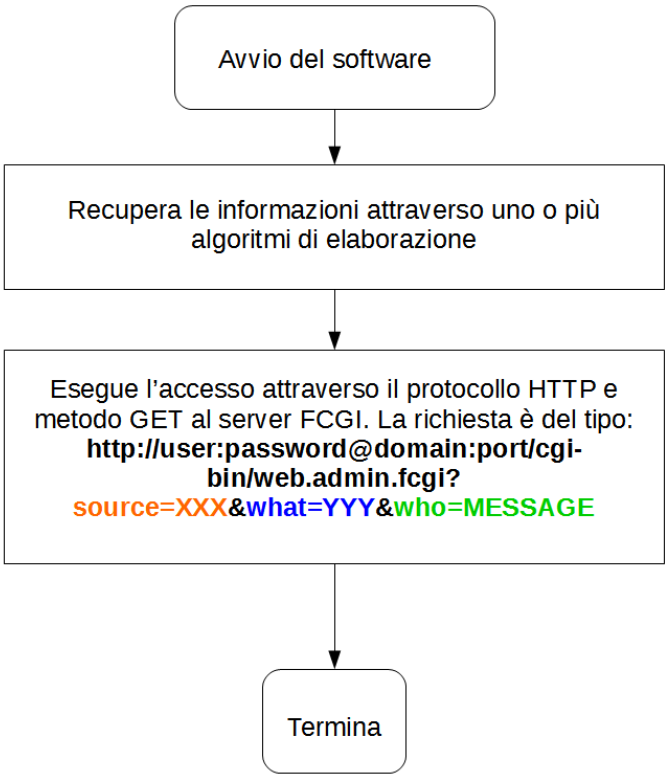

Figura 19. Diagramma di flusso tipico per un processo diagnostico lato client.

7.3 Installazione e descrizione clients amministrativi

Nelle appendici A7 e A8 sono riportati i sorgenti di un modulo attuatore. Il modulo, il cui diagramma di flusso è riportato nella figura 20, si occupa di eseguire il reset del *server* su cui viene installato. Tale software funziona da *daemon* e quindi nella veste proposta e di diretto utilizzo nei sistemi *UNIX-like*. Nulla vieta di utilizzarlo su altri sistemi operativi utilizzando le diverse *API* (Application Programming Interface) che i diversi modelli mettono a disposizione per la programmazione dei servizi. Il principio di funzionamento degli attuatori è lo stesso dei *clients* diagnostici (figura 19): si inoltra una semplice richiesta di tipo *GET* attraverso protocollo *HTTP* al *server web* che richiamerà l'estensione *FCGI*. La pagina *web* richiesta conterrà semplicemente il valore *vero* o *falso* che indica al server la necessità di riavviarsi. La sicurezza dell'operazione è garantita attraverso la protezione delle singole pagine *web* che ogni *server web* mette a disposizione. Ciò evita che si possa resettare accidentalmente un server in seguito a un accesso involontario e pubblico, per esempio di un motore di ricerca. Per evitare che dalle pagine *web* dell'*RDAT* un utente riavvii involontariamente un server viene richiesta una conferma attraverso una finestra di dialogo *javascript*.

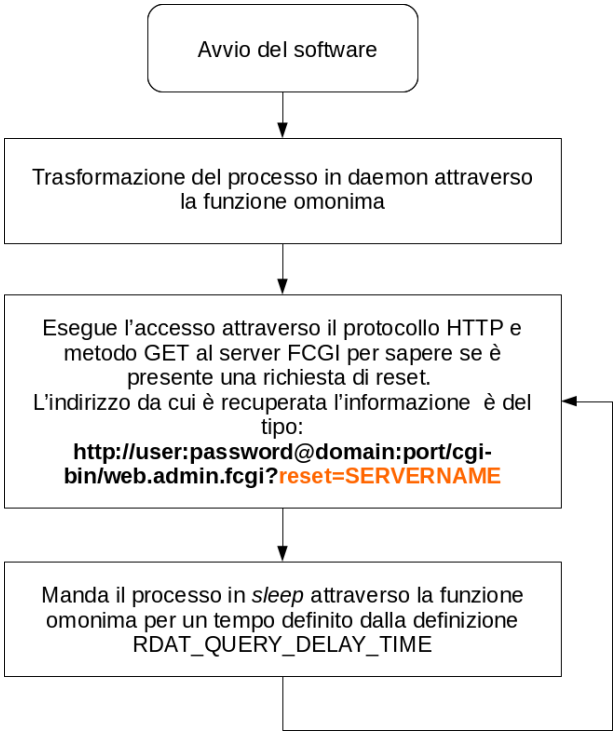

Figura 20. Diagramma di flusso di un client amministrativo tipo (reset server).

Conclusioni

Il problema della manutenzione dei sistemi di monitoraggio è importante per la continuità dell'acquisizione. Un sistema che processa i segnali diagnostici e formula automaticamente scenari riguardanti i possibili interventi, può aiutare a risparmiare tempo e incrementare la precisione nella risoluzione dei problemi. Il corredo di strumenti di amministrazione automatici e manuale permette di intervenire, anche in condizioni sfavorevoli, per ripristinare le funzionalità compromesse.

Ringraziamenti

Volevamo ringraziare tutta la Segreteria di Redazione del CEN che si è dimostrata, ancora una volta, molto veloce ed efficiente. Un ringraziamento particolare è dovuto alla dott.ssa Rossella Celi per la cordialità e la disponibilità mostrata.

Bibliografia

Del Negro C., Napoli R., Sicali A., (2002). *Automated system for magnetic monitoring of active volcanoes*. Bull. Volcanol. 64, 94-99.

Sicali A., Bonaccorso A., (2013). *Gestione dei dilatometri installati in pozzi profondi all'Etna*. Rapporti Tecnici INGV n°. 258, ISSN 2039-7941.

Currenti G., Napoli R., Sicali A., Greco F., Del Negro C., (2014). *GEOFIM: a WebGIS application for integrated geophysical modelling in active volcanic regions.* Computers & Geosciences, 70, 120-127, http://dx.doi.org/10.1016/j.

Sicali A., Amantia A., Cappuccio P., (2016). *Linee guida e criticità nella progettazione di sistemi per l'acquisizione di dati geofisici in prossimità di vulcani attivi.* Rapporti Tecnici INGV n°. 347, ISSN 2039-7941.

Sicali A., Amantia A., Cappuccio P., (2017). *Sistema di DNS dinamico utilizzante la messaggistica SMS.* Rapporti tecnici INGV n°. 367, ISSN 2039-7941.

Sitografia

Wikipedia, Self-Monitoring, Analysis and Reporting Technology, https://it.wikipedia.org/wiki/Self-Monitoring,_Analysis_and_Reporting_Technology.
Wikipedia, FastCGI, https://en.wikipedia.org/wiki/FastCGI.
Wikipedia, Hypertext Transfer Protocol, https://it.wikipedia.org/wiki/Hypertext_Transfer_Protocol.
Wikipedia, Comma-separated values, https://it.wikipedia.org/wiki/Comma-separated_values.
Wikipedia, HTML, https://it.wikipedia.org/wiki/HTML.
Wikipedia, CSS, https://it.wikipedia.org/wiki/CSS.
Wikipedia, Common Gateway Interface, https://it.wikipedia.org/wiki/Common_Gateway_Interface.
Wikipedia, Stack Buffer Overflow, https://en.wikipedia.org/wiki/Stack_buffer_overflow.
Wikipedia, Pattern Recognition, https://en.wikipedia.org/wiki/Pattern_recognition.
Wikipedia, Cardinality, https://en.wikipedia.org/wiki/Cardinality.
Wikipedia, PID Controller, https://en.wikipedia.org/wiki/PID_controller.
Wikipedia, Event-driven programming, https://en.wikipedia.org/wiki/Event-driven_programming.
Wikipedia, Mobile app, https://en.wikipedia.org/wiki/Mobile_app.
Wikipedia, Multi touch, https://en.wikipedia.org/wiki/Multi-touch.
Wikipedia, JQuery, https://it.wikipedia.org/wiki/JQuery.
Wikipedia, JQuery UI, https://it.wikipedia.org/wiki/JQuery_UI.
Wikipedia, Scalability, https://en.wikipedia.org/wiki/Scalability.
Wikipedia, Expert System, https://en.wikipedia.org/wiki/Expert_system.
Flot, www.flotcharts.org.

Appendice A1. Codice sorgente: web.admin.h

```
#ifndef _WEB_ADMIN_H
#define _WEB_ADMIN_H

#define MAX_QUERY_SIZE 4096
#define MAX_PARAMETERS 1000
#define MAX_INSERTS    1000000

#define MAX_SHOWED_INSERTS 100

#define WHAT_TEXT "what"
#define SOURCE_TEXT "source"
#define WHO_TEXT "who"

#define SERVERS_TEXT        "servers"
#define INSERTS_TEXT        "inserts"
#define RESET_TEXT          "reset"
#define BOOTING_TEXT        "booting"
#define IP_TEXT             "ip"
#define DOWNLOAD_DELAY_TEXT "download_delay"
#define LINK_TEXT           "link"
#define GOOD_TEXT           "Good"
#define FAIL_TEXT           "Fail"
#define OK_TEXT             "OK"
#define DDNS_TEXT           "ddns"
#define IDLE_TEXT           "idle"
#define DDNS_CREG           "creg"
#define DDNS_COPS           "cops"
#define DDNS_CSQ            "csq"
#define SIGNAL_TEXT         "signal"
#define DETAILS_TEXT        "details"
#define SIGNALS_TEXT        "signals"
#define SYNTESIS_TEXT       "syntesis"
#define LAST_DATA           "last_data"

#define POWER_FAILURE_STATUS       "power_failure"
#define ACQUISITION_FAILURE_STATUS "acquisition_failure"
#define SIGNAL_FAILURE_STATUS      "signal_failure"
#define SIGNAL_LOW_STATUS          "signal_low"
#define GOOD_STATUS                "good"
#define LINK_FAILURE_STATUS        "link_failure"

#define POWER_FAILURE       "ATTENTION!!! the station may be me down"
#define ACQUISITION_FAILURE "ATTENTION!!! the acquisition may be halted"
#define ROUTER_FAILURE      "Router, Power Supply, Antenna broke, "
#define WWW_FAILURE         "Shoebox Web Server"
#define DAP24_FAILURE       "Dap24, "
#define DOWNLOAD_FAILURE    "syncStrain (SSH key absent), "
#define SHOEBOX_FAILURE     "Shoebox, "
#define IBOOT_FAILURE       "iBoot, "
#define SSH_FAILURE         "SSH, "
#define RABBIT_FAILURE      "Rabbit, "
#define ROUTER_WEB_FAILURE  "Router Web Server, "
#define LOW_RADIO_SIGNAL    "Low Radio Signal, "

#define MAIL_FILE    "/var/log/web.admin/mail"
#define LOG_FILE     "/var/log/web.admin/web.admin.log"
#define INSERTS_FILE "/var/log/web.admin/inserts.log"

#define MAX_SERVERS  10
#define MAX_STATIONS 50
```

```
#define MAX_DAYS 2

//#define SIMPLE_TEST

#define NMONTHS 12
#define NHOURS  24

#define MAX_RESET_HOURS 12

#define DDNS_DELAY 10 // minutes
#define DDNS_SERVER "ctdyndns"
#define MAX_DDNS_FAULTS 10

typedef struct DT {
        int day;
        int month;
        int year;
        int hour;
        int min;
        int sec;
} DT;

typedef struct PARAM {
        char name[64];
        char value[64];
} PARAM;

typedef struct SERVER_FIELD {
        char name[64];
        bool reset;
} SERVER_FIELD;

#define GREEN  0
#define YELLOW 1
#define RED    2

typedef struct STATIONS_FIELD {
        char name[64];
        char ip[64];
        DT ip_timestamp;
        DT ip_dt;
        int delay;
        DT delay_dt;
        int router;
        int shoebox;
        int iboot;
        int rabbit;
        int ssh;
        DT signal_timestamp;
        DT last_data;
        char signal[64];
        char diagnosis[512];
        char alert[512];
        int status;
        char general_status[32];
} STATIONS_FIELD;

#define INSERT_RECORDS 5
typedef struct INSERT_FIELD {
        char date[32];
        char time[32];
```

```
        char source[64];
        char what[64];
        char who[64];
} INSERT_FIELD;

void checkDDNS(bool syntesis=false);

#endif //_WEB_ADMIN_H
```

Appendice A2. Codice sorgente: web.admin.cpp

```
#include <fcgi_stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include "web.admin.h"
#include <time.h>

INSERT_FIELD inserts[MAX_INSERTS];
int ninserts=0;

SERVER_FIELD servers[MAX_SERVERS];
int nservers=0;

STATIONS_FIELD stations[MAX_STATIONS];
int nstations=0;

DT ddns_dt;
DT ddns_dt_test;
char ddns_csq[64]="";
char ddns_creg[64]="";
char ddns_cops[64]="";

const char *mails[]={"antonino.sicali@ingv.it",NULL};

int CmpTime(const DT &a,const DT &b)
{
         if (a.year < b.year) return -1;
         if (a.year > b.year) return 1;
         if (a.month < b.month) return -1;
         if (a.month > b.month) return 1;
         if (a.day < b.day) return -1;
         if (a.day > b.day) return 1;
         if (a.hour < b.hour) return -1;
         if (a.hour > b.hour) return 1;
         if (a.min < b.min) return -1;
         if (a.min > b.min) return 1;
         if (a.sec < b.sec) return -1;
         if (a.sec > b.sec) return 1;

        return 0;
}

bool operator>(const DT &a,const DT &b)
{
        return CmpTime(a,b) > 0;
}

void SaveLog(const char *s=NULL)
{
        time_t t;
        struct tm *tt;
        char d[128];

        t=time(0);
        tt = localtime(&t);
        strftime(d,sizeof(d)-1,"%A %d/%m/%y %T %Z",tt);

        FILE *drain=fopen(LOG_FILE,"a+");
        if (!s) fprintf(drain,"%s %s %s %s\n",d,getenv("HTTP_USER_AGENT"),
                        getenv("REMOTE_ADDR"),
```

```
                        getenv("REQUEST_URI"));
                        else
                        fprintf(drain,"%s %s\n",d,s);
        fclose(drain);
}

void getDT(DT &dt,char *d,char *t)
{
    int r[3];

    memset(&dt,0,sizeof(dt));

    if (sscanf(d,"%d/%d/%d",r,r+1,r+2) != 3) return;
    dt.day=r[0];
    dt.month=r[1];
    dt.year=r[2];
    dt.year+=2000;
    if (sscanf(t,"%d:%d:%d",r,r+1,r+2) != 3) return;
    dt.hour=r[0];
    dt.min=r[1];
    dt.sec=r[2];
}

void getTimestamp(DT &dt,char *timestamp)
{
    memset(&dt,0,sizeof(dt));

    dt.year=(timestamp[0]-'0')*1000+(timestamp[1]-'0')*100+(timestamp[2]-
     '0')*10+timestamp[3]-'0';
    dt.month=(timestamp[4]-'0')*10+timestamp[5]-'0';
    dt.day=(timestamp[6]-'0')*10+timestamp[7]-'0';
    dt.hour=(timestamp[8]-'0')*10+timestamp[9]-'0';
    dt.min=(timestamp[10]-'0')*10+timestamp[11]-'0';
    dt.sec=(timestamp[12]-'0')*10+timestamp[13]-'0';
}

int getMonthDays(int mon,int year)
{
        int days[NMONTHS]={31,0,31,30,31,30,31,31,30,31,30,31};
        int v;

        v=days[mon-1];
        if (!v) v=year%4?28:29;
        return v;
}

int getDays(int year,int start,int stop)
{
    int i;
    int l=0;

    for (i=start;i<=stop;i++) l += getMonthDays(i,year);
    return l;
}

int calculateHours(const DT &a,const DT &b)
{
    int i;
    int l=0;

    if (b.year-a.year < 0) return 0;
```

```
    for (i=0;i<=b.year-a.year;i++) l += getDays(i+a.year,1,NMONTHS);

    if (a.month > 1) l -= getDays(a.year,1,a.month-1);
    if (b.month < NMONTHS) l -= getDays(b.year,b.month+1,NMONTHS);

    l -= a.day-1;
    l -= getMonthDays(b.month,b.year)-b.day;

    l *= NHOURS;
    l -= a.hour;
    l -= NHOURS-1-b.hour;

    return l;
}


void sendMail(const STATIONS_FIELD &station)
{
        #ifdef EMAIL
        char cmd[100];
        char subject[100];


        sprintf(subject,"\"%s - %s\"",station.name,station.diagnosis);

         FILE *handle = fopen(MAIL_FILE,"wt+");
        fprintf(handle,"Diagnosis: %s\n",station.diagnosis);
        fprintf(handle,"Name: %s\n",station.name);
        fprintf(handle,"IP:                                                    %s
         (%2.2d/%2.2d/%4.4d %2.2d:%2.2d:%2.2d)\n",station.ip,station.ip_dt.day,st
         ation.ip_dt.month,station.ip_dt.year,station.ip_dt.hour,station.ip_dt.mi
         n,station.ip_dt.sec);
        fprintf(handle,"Download delay (days): %d\n",station.delay);
        fprintf(handle,"Router:  %s\n",station.router?FAIL_TEXT:OK_TEXT);
        fprintf(handle,"Shoebox: %s\n",station.shoebox?FAIL_TEXT:OK_TEXT);
        fprintf(handle,"iBoot: %s\n",station.iboot?FAIL_TEXT:OK_TEXT);
        fclose(handle);

        int i=0;
        while(mails[i]) {
                sprintf(cmd,"/usr/bin/mailx     -u     rdat     -s     %s     %s
                < %s",subject,mails[i],MAIL_FILE);
                system(cmd);
                printf(cmd);
                i++;
        }
        #endif
}

void getTime(DT &dt)
{
        time_t t;
        struct tm *tt;

        t=time(0);
        tt = localtime(&t);

        dt.year=tt->tm_year+1900;
        dt.day=tt->tm_mday;
        dt.month=tt->tm_mon+1;
        dt.hour=tt->tm_hour;
        dt.min=tt->tm_min;
```

```
        dt.sec=tt->tm_sec;

}

void Diagnosis(STATIONS_FIELD &station,bool mail)
{
        char old[512];
        DT dt;

        getTime(dt);

        int ip_delay=!station.signal_timestamp.day?
                MAX_RESET_HOURS:calculateHours(station.signal_timestamp,dt);

        strcpy(old,station.diagnosis);
        station.diagnosis[0]=0;
        station.alert[0]=0;
        station.general_status[0]=0;
        if (!station.ip[0]) return;

        station.status=YELLOW;

                if   (ip_delay   >=   MAX_RESET_HOURS   ||   (station.router   &&
        station.shoebox && station.iboot && station.ssh && station.rabbit)) {
                                                                        if
        (ip_delay >= MAX_RESET_HOURS) { strcpy(station.diagnosis,ROUTER_FAILURE);
        strcpy(station.alert,POWER_FAILURE);
        strcpy(station.general_status,POWER_FAILURE_STATUS);
        station.status=RED; }

        else

        {

        strcat(station.diagnosis,LOW_RADIO_SIGNAL);

        if  (station.delay)  strcpy(station.general_status,SIGNAL_FAILURE_STATUS);
        else strcpy(station.general_status,SIGNAL_LOW_STATUS);

              }
                                                                        if    (mail)
        sendMail(station);
                                                                        int
        l=strlen(station.diagnosis);
                                                                        if
        (station.diagnosis[l-2] == ',') station.diagnosis[l-2]=0;
                                                                        return;
        }
        if (station.router) strcat(station.diagnosis,ROUTER_WEB_FAILURE);
        if (station.iboot) strcat(station.diagnosis,IBOOT_FAILURE);
        if (station.ssh) strcat(station.diagnosis,SSH_FAILURE);
        if (station.rabbit) strcat(station.diagnosis,RABBIT_FAILURE);
        if (station.delay >= MAX_DAYS) {
                        if                                      (station.shoebox)
      { strcat(station.diagnosis,SHOEBOX_FAILURE); }
                                else
                                {
                                strcat(station.diagnosis,DAP24_FAILURE);
                                strcat(station.diagnosis,", ");
                                strcat(station.diagnosis,DOWNLOAD_FAILURE);
                                }
                strcpy(station.alert,ACQUISITION_FAILURE);
```

```
                strcpy(station.general_status,ACQUISITION_FAILURE_STATUS);
        }
        else
        {
                        if                                              (station.shoebox)
         {                                  strcat(station.diagnosis,WWW_FAILURE);
         strcpy(station.general_status,LINK_FAILURE_STATUS); }
                                            else
                                            if              (!station.diagnosis[0])
         {                                   strcpy(station.diagnosis,GOOD_TEXT);
         strcpy(station.general_status,GOOD_STATUS); station.status=GREEN; }
                }

        int l=strlen(station.diagnosis);
        if (station.diagnosis[l-2] == ',') station.diagnosis[l-2]=0;

        if (mail && strcmp(old,station.diagnosis)) sendMail(station);
}

void setReset(const char *name,bool reset)
{
        int i;

        for (i=0;i<nservers;i++) if (!strcmp(servers[i].name,name)) {
                servers[i].reset=reset;
                return;
        }
        if (nservers == MAX_SERVERS) return;
        strcpy(servers[nservers].name,name);
        servers[nservers].reset=reset;
        nservers++;
}

void setDelay(char *s,bool mail)
{
        int i=0;
        int delay;
        DT dt;

        while(s[i] && s[i] != ':') i++;

        if (!s[0]) return;

        s[i]=0;
        i++;
        delay=i;
        while(s[i] && s[i] != ':') i++;
        if (sscanf(s+delay,"%d",&delay) != 1) return;

        getTimestamp(dt,s+i+1);

        for (i=0;i<nstations;i++) if (!strcmp(stations[i].name,s)) {
                stations[i].delay=delay;
                stations[i].delay_dt=dt;
                Diagnosis(stations[i],mail);
                return;
        }
        if (nstations == MAX_STATIONS) return;
        strcpy(stations[nstations].name,s);
        stations[nstations].delay=delay;
        stations[nstations].delay_dt=dt;
        Diagnosis(stations[nstations],mail);
```

```
        nstations++;
}

void setLastData(char *s)
{
        int i=0;
        int delay;
        DT dt;

        while(s[i] && s[i] != ':') i++;

        if (!s[0]) return;

        s[i]=0;
        getTimestamp(dt,s+i+1);

        for (i=0;i<nstations;i++) if (!strcmp(stations[i].name,s)) {
                stations[i].last_data=dt;
                return;
        }
        if (nstations == MAX_STATIONS) return;
        strcpy(stations[nstations].name,s);
        stations[nstations].last_data=dt;
        nstations++;
}

void setLink(char *s,bool mail)
{
        int i=0;
        int r[5];

        while(s[i] && s[i] != ':') i++;

        if (!s[0]) return;

        s[i]=0;

        if (sscanf(s+i+1,"%d,%d,%d,%d,%d",r,r+1,r+2,r+3,r+4) != 5) return;


        for (i=0;i<nstations;i++) if (!strcmp(stations[i].name,s)) {
                stations[i].router=r[0];
                stations[i].shoebox=r[1];
                stations[i].iboot=r[2];
                stations[i].rabbit=r[3];
                stations[i].ssh=r[4];
                Diagnosis(stations[i],mail);
                return;
        }
        if (nstations == MAX_STATIONS) return;
        strcpy(stations[nstations].name,s);
        stations[nstations].router=r[0];
        stations[nstations].shoebox=r[1];
        stations[nstations].iboot=r[2];
        stations[nstations].rabbit=r[3];
        stations[nstations].ssh=r[4];
        Diagnosis(stations[nstations],mail);
        nstations++;
}

void addDelay(const DT &a,DT &b)
{
```

```
        b=a;
        b.min = a.min+DDNS_DELAY;
        if (b.min >= 60) {
                b.min-=60;
                b.hour++;
                if (b.hour >= 24) {
                        b.hour -= 24;
                        b.day++;
                        int days=getMonthDays(b.month,b.year);
                        if (b.day > days) {
                                b.day -= days;
                                b.month++;
                                if (b.month > 12) {
                                        b.month -= 12;
                                        b.year++;
                                }
                        }
                }
        }
}

void setDDNS(char *s,char *date,char *time)
{
        if (!strcmp(s,IDLE_TEXT)) {
                        getDT(ddns_dt,date,time);
                        addDelay(ddns_dt,ddns_dt_test);
        }
}

void setSignal(char *s,char *date,char *time)
{
        int i=0;
        int signal;
        int timestamp;

        while(s[i] && s[i] != ':') i++;

        if (!s[0]) return;

        s[i]=0;
        i++;
        signal=i;

        while(s[i] && s[i] != ':') i++;

        if (!s[0]) return;

        s[i]=0;
        i++;
        timestamp=i;


        for (i=0;i<nstations;i++) if (!strcmp(stations[i].name,s)) {
                strcpy(stations[i].signal,s+signal);
                getTimestamp(stations[i].signal_timestamp,s+timestamp);
                return;
        }
        if (nstations == MAX_STATIONS) return;
        strcpy(stations[nstations].name,s);
        strcpy(stations[nstations].signal,s+signal);
        getTimestamp(stations[nstations].signal_timestamp,s+timestamp);
        nstations++;
```

```
}

void setIP(char *s,char *date,char *time)
{
        int i=0;
        int ip;
        int timestamp;

        while(s[i] && s[i] != ':') i++;

        if (!s[0]) return;

        s[i]=0;
        i++;
        ip=i;
        SaveLog(s+ip);

        while(s[i] && s[i] != ':') i++;

        if (!s[0]) return;

        s[i]=0;
        i++;
        timestamp=i;
        SaveLog(s+timestamp);


        for (i=0;i<nstations;i++) if (!strcmp(stations[i].name,s)) {
                SaveLog("Found station");
                strcpy(stations[i].ip,s+ip);
                SaveLog("get timestamp");
                getTimestamp(stations[i].ip_timestamp,s+timestamp);
                SaveLog("get ddns time");
                getDT(stations[i].ip_dt,date,time);
                SaveLog("Return");
                return;
        }
        if (nstations == MAX_STATIONS) return;
        SaveLog("New station");
        strcpy(stations[nstations].name,s);
        strcpy(stations[nstations].ip,s+ip);
        getTimestamp(stations[nstations].ip_timestamp,s+timestamp);
        getDT(stations[nstations].ip_dt,date,time);
        nstations++;
}

bool getReset(char *name)
{
        int i;

        static int n=0;

        if (!strcmp(name,DDNS_SERVER)) {
                DT dt;
                getTime(dt);

                if (dt > ddns_dt_test) n++; else n=0;
                if (n == MAX_DDNS_FAULTS) { n=0; setReset(DDNS_SERVER,true); }
        }

        for (i=0;i<nservers;i++) if (!strcmp(servers[i].name,name))
                                        return servers[i].reset;
```

```
        return false;
}

void trim(char *line)
{
        int j=0,i;

        if (!line[0]) return;

        i=0;
        while(line[i]) {
                                if (line[i] == '\t') line[i]=0x20;
                                i++;
        }

        i=0;
        while(line[i] && (line[i] <= 0x20 || line[i] >= 0x80)) i++;
        //while(line[i] == 0x20 || line[i] == 0x0a || line[i] == 0x09) i++;
        if (!line[i]) { line[0]=0; return; }

        while(line[j+i]) {
                line[j]=line[j+i];
                j++;
        }

        line[j]=0;

        while(line[j] <= 0x20 || line[j] >= 0x80) j--;
        line[j+1]=0;
}

void processInsert(INSERT_FIELD &insert,bool mail)
{
        char tmp[64];

        strcpy(tmp,insert.who);

        if (!strcmp(insert.what,RESET_TEXT)) { setReset(tmp,true); return; }
                if                                (!strcmp(insert.what,BOOTING_TEXT))
        { setReset(insert.source,false); return; }
                if                                      (!strcmp(insert.what,IP_TEXT))
        { setIP(tmp,insert.date,insert.time); return; }
                if                                 (!strcmp(insert.what,SIGNAL_TEXT))
        { setSignal(tmp,insert.date,insert.time); return; }
                if                         (!strcmp(insert.what,DOWNLOAD_DELAY_TEXT))
        { setDelay(tmp,mail); return; }
                if     (!strcmp(insert.what,LINK_TEXT))     {     setLink(tmp,mail);
        return; }
                if                                       (!strcmp(insert.what,DDNS_TEXT))
        { setDDNS(tmp,insert.date,insert.time); return; }
                if    (!strcmp(insert.what,DDNS_COPS))    {    strcpy(ddns_cops,tmp);
        return; }
                if     (!strcmp(insert.what,DDNS_CSQ))     {     strcpy(ddns_csq,tmp);
        return; }
                if    (!strcmp(insert.what,DDNS_CREG))    {    strcpy(ddns_creg,tmp);
        return; }
                    if    (!strcmp(insert.what,LAST_DATA))    {    setLastData(tmp);
        return; }
             }

void parserInsert(char *s)
{
```

```
        int i=0;
        int last=0;
        int n=0;

        trim(s);

        while(s[i]) {
                if (s[i] == 0x20) {
                        s[i]=0;
                        trim(s+last);
                        switch(n) {
                                case 0:
                                        strcpy(inserts[ninserts].date,s+last);
                                        break;
                                case 1:
                                        strcpy(inserts[ninserts].time,s+last);
                                        break;
                                case 2:
                                        strcpy(inserts[ninserts].source,s+last);
                                        break;
                                case 3:
                                        strcpy(inserts[ninserts].what,s+last);
                                        break;
                                case 4:
                                        strcpy(inserts[ninserts].who,s+last);
                                        break;
                        }
                        n++;
                        i++;
                        last=i;
                        continue;
                }
                i++;
        }

        if   (n   ==   INSERT_RECORDS-1)   {   strcpy(inserts[ninserts].who,s+last);
n++; }
        if   (n   ==   INSERT_RECORDS)   {   processInsert(inserts[ninserts],false);
ninserts++; }
}

void loadInserts(void)
{
        char s[128];

        FILE *handle=fopen(INSERTS_FILE,"rt");
        if (!handle) return;
        while(fgets(s,sizeof(s)-1,handle)) {
                SaveLog(s);
                parserInsert(s);
        }
        fclose(handle);

        int i;
        for  (i=0;i<nstations;i++)  if  (strcmp(stations[i].diagnosis,GOOD_TEXT))
sendMail(stations[i]);
}

bool insert(PARAM *parameters,int n)
{
        int i;
        time_t t;
```

```
        struct tm *tt;
        char ds[16];
        char ts[16];
        int source=-1,what=-1,who=-1;

        t=time(0);
        tt = localtime(&t);
        strftime(ds,sizeof(ds)-1,"%d/%m/%y",tt);
        strftime(ts,sizeof(ts)-1,"%T",tt);

                for (i=0;i<n;i++) {
                        if    (!strcmp(parameters[i].name,SOURCE_TEXT))    {    if
        (source >= 0) return false; else source=i; }
                        if (!strcmp(parameters[i].name,WHAT_TEXT)) { if (what >=
        0) return false; else what=i; }
                        if (!strcmp(parameters[i].name,WHO_TEXT)) { if (who >= 0)
        return false; else who=i; }
                }

        if (source < 0 || what < 0 || who < 0) return false;

        FILE *drain=fopen(INSERTS_FILE,"a+");
                fprintf(drain,"%s %s %s %s %s\n",ds,ts,parameters[source].value,p
        arameters[what].value,parameters[who].value);
        fclose(drain);

        strcpy(inserts[ninserts].time,ts);
        strcpy(inserts[ninserts].date,ds);
        strcpy(inserts[ninserts].source,parameters[source].value);
        strcpy(inserts[ninserts].what,parameters[what].value);
        strcpy(inserts[ninserts].who,parameters[who].value);
        processInsert(inserts[ninserts],true);
        ninserts++;

        return true;
}

bool getParameters(char *s,PARAM *parameters,int &n,int nmax)
{
        int i=0;
        int last;

        while(s[i] && s[i] != '?') i++;

        if (!s[i]) return false;

        i++;

        last=i;
        while(s[i]) {
                if (s[i] == '&') {
                        s[i]=0;
                        strcpy(parameters[n].value,s+last);
                        n++;
                        if (n == nmax) return false;
                        i++;
                        last=i;
                }
                if (s[i] == '=') {
                        s[i]=0;
                        strcpy(parameters[n].name,s+last);
                        i++;
```

```
                        last=i;
                }
                i++;
        }

        strcpy(parameters[n].value,s+last);
        n++;

        return true;
}

char query[MAX_QUERY_SIZE];
PARAM parameters[MAX_PARAMETERS];

void checkDDNS(bool syntesis)
{
        char tclass[64];
        char mclass[64];
        DT dt;

        getTime(dt);

        if (syntesis) {
                int r=0;
                if (dt > ddns_dt_test) r++;
                if (!strcmp(ddns_cops,"0,0")) r++;
                if (r) printf("DDNS Failure\n"); else printf("DDNS OK\n");
                switch(r) {
                        case 0:
                                printf("green\n");
                                break;
                        case 1:
                                printf("yellow\n");
                                break;
                        case 2:
                                printf("red\n");
                                break;
                }
                return;
        }

                if (dt > ddns_dt_test) strcpy(tclass,"station_red"); else
        strcpy(tclass,"station_green");
                if (!strcmp(ddns_cops,"0,0")) strcpy(mclass,"station_red"); else
        strcpy(mclass,"station_green");
                printf("<h1 class=\"%s\">Last DDNS check
        at %2.2d/%2.2d/%4.4d %2.2d:%2.2d:%2.2d</h1><br><h1 class=\"%s\">DDNS
        modem signal: %s, operator: %s, registration: %s</h1>",

        tclass,ddns_dt.day,ddns_dt.month,ddns_dt.year,ddns_dt.hour,ddns_dt.min,d
        dns_dt.sec,mclass,ddns_csq,ddns_cops,ddns_creg);
}

void showSignals(char *s)
{
        int i;
        char name[MAX_STATIONS][16];
        char param[MAX_STATIONS][16];
        bool correlation=false;
        const char *params[]={"s1a","bar24","s2a","tmp","s1s","pwr",NULL};

        memset(name,0,sizeof(name));
```

```
memset(param,0,sizeof(param));

correlation=!strcmp(s,"correlation");

        if (!strcmp(s,"pwr") || !strcmp(s,"slope") || !strcmp(s,"tides")
|| !strcmp(s,"correlation")) {
        for (i=0;i<nstations;i++) {
                strcpy(&name[i][0],stations[i].name);
                strcpy(&param[i][0],s);
        }
                }
                else
                {
        i=0;
        while(params[i]) {
                        strcpy(&name[i][0],s);
                        strcpy(&param[i][0],params[i]);
                        i++;
        }
                }

printf("Content-type: text/html\r\n\r\n");
printf("<html>\n");
printf("<head>\n");
printf("<meta charset=\"UTF-8\">\n");
        printf("<link        rel=\"stylesheet\"         type=\"text/css\"
href=\"/web.admin.css\">\n");
        printf("<script  language=\"javascript\"  type=\"text/javascript\"
src=\"/flot/jquery.js\"></script>\n");
        printf("<script  language=\"javascript\"  type=\"text/javascript\"
src=\"/flot/jquery.flot.js\"></script>\n");
        printf("<script  language=\"javascript\"  type=\"text/javascript\"
src=\"/flot/jquery.flot.time.js\"></script>\n");
        printf("<script  language=\"javascript\"  type=\"text/javascript\"
src=\"/flot/jquery.flot.selection.js\"></script>\n");
        printf("<script  language=\"javascript\"  type=\"text/javascript\"
src=\"/flot/jquery.flot.navigate.js\"></script>\n");
        printf("<link                                    rel=\"stylesheet\"
href=\"/jquery/themes/base/jquery-ui.css\">\n");
printf("<script src=\"/jquery/jquery-ui.js\"></script>\n");
i=0;
while(param[i][0]) {
                printf("<script                    language=\"javascript\"
type=\"text/javascript\"
src=\"/signals/%s.%s.js\"></script>\n",name[i],param[i]);
                i++;
        }

printf("<script type=\"text/javascript\">\n");

        printf("$(  function()  {\n$(  \"input\"  ).checkboxradio({\nicon:
false\n});\n} );\n");

i=0;
while(param[i][0]) {
printf("$(function() {\n");
printf("var options = {\n");
if (correlation) {
                printf("xaxis:   {\nticks:   10,\npanRange:   true,\nmin:
10.5,\nmax: 15.0},\n");
                printf("yaxis:   {\nticks:   10,\npanRange:   true,\nmin:
10.5,\nmax: 15.0},\n");
```

```
                printf("grid: { hoverable: true,  autoHighlight: true,
markings: [\
                        { xaxis: { from: 10.5, to: 12.0 }, yaxis: { from:
12.5, to: 15.0 } ,color: \"#ff5c33\"},\
                        { xaxis: { from: 12.5, to: 15.0 }, yaxis: { from:
12.5, to: 15.0 } ,color: \"#5cd65c\"},\
                        { xaxis: { from: 10.5, to: 12.0 }, yaxis: { from:
10.5, to: 12.0 } ,color: \"#e6e600\"}]},");
                printf("points: { show: true },\n");
        } else
        {
        printf("xaxis:    {\nminTickSize:    [1,    \"second\"],\nmode:
\"time\",\ntimeformat:\"%%d/%%m/%%y %%H:%%M:%%S\",\nticks: 10,\npanRange:
true\n},\n");
        printf("yaxis: {\nticks: 10,\npanRange: true\n},\n");
        }
        printf("selection: {\nmode: \"y\"\n},\n");
        printf("pan: {\ninteractive: false\n}\n");
        printf("};\n");

        printf("var   plot   =   $.plot(\"#signal_%s_%s\",   [%s_%s],
options);\n",name[i],param[i],name[i],param[i]);
        printf("var pan=false;\n");

        printf("$(\"#signal_%s_%s\").bind(\"plotselected\",     function
(event, ranges) {\n",name[i],param[i]);

        printf("$.each(plot.getYAxes(), function(_, axis) {\nvar opts =
axis.options;\nopts.min      =      ranges.yaxis.from;\nopts.max      =
ranges.yaxis.to;\n});\n");
        printf("$.each(plot.getXAxes(), function(_, axis) {\nvar opts =
axis.options;\nopts.min      =      ranges.xaxis.from;\nopts.max      =
ranges.xaxis.to;\n});\n");
        printf("plot.setupGrid();\n");
        printf("plot.draw();\n");
        printf("plot.clearSelection();\n");
        printf("});");

        printf("$(\"#footer\").prepend(\"Flot \" + $.plot.version + \"
&ndash; \");\n");

        printf("$(document).ready(function() {\n");

 printf("$('input[type=radio][name=signal_bar_%s_%s]').change(function()
{\n",name[i],param[i]);
        printf("var options=plot.getOptions();\n");
        printf("var s=options['selection'];\n");
        printf("var p=options['pan'];\n");
        printf("var xa=options['xaxis'];\n");
        printf("var ya=options['yaxis'];\n");
             printf("if (this.value == 'zoomx') {\n");
        printf("s.mode=\"x\";\n");
        printf("if  (pan)  {  pan=false;  p.interactive=false;  plot  =
$.plot(\"#signal_%s_%s\",                                    [%s_%s],
options); }",name[i],param[i],name[i],param[i]);
             printf("} else\n");
        printf("if (this.value == 'zoomxy') {\n");
        printf("s.mode=\"xy\";\n");
        printf("if  (pan)  {  pan=false;  p.interactive=false;  plot  =
$.plot(\"#signal_%s_%s\",                                    [%s_%s],
options); }",name[i],param[i],name[i],param[i]);
             printf("} else\n");
```

```
        printf("if (this.value == 'panx') {\n");
        printf("p.interactive=true;\n");
        printf("s.mode=null\n");
        printf("$.each(plot.getYAxes(), function(_, axis) {\nvar opts =
axis.options;\nopts.panRange = false;\n});\n");
        printf("$.each(plot.getXAxes(), function(_, axis) {\nvar opts =
axis.options;\nopts.panRange = true;\n});\n");
        printf("if  (!pan)  {  pan=true;  p.interactive=true;  plot  =
$.plot(\"#signal_%s_%s\",                                          [%s_%s],
options); }",name[i],param[i],name[i],param[i]);
        printf("} else\n");
        printf("if (this.value == 'pany') {\n");
        printf("p.interactive=true;\n");
        printf("s.mode=null\n");
        printf("$.each(plot.getYAxes(), function(_, axis) {\nvar opts =
axis.options;\nopts.panRange = true;\n});\n");
        printf("$.each(plot.getXAxes(), function(_, axis) {\nvar opts =
axis.options;\nopts.panRange = false;\n});\n");
        printf("if  (!pan)  {  pan=true;  p.interactive=true;  plot  =
$.plot(\"#signal_%s_%s\",                                          [%s_%s],
options); }",name[i],param[i],name[i],param[i]);
        printf("} else\n");
        printf("if (this.value == 'panxy') {\n");
        printf("p.interactive=true;\n");
        printf("s.mode=null\n");
        printf("$.each(plot.getYAxes(), function(_, axis) {\nvar opts =
axis.options;\nopts.panRange = true;\n});\n");
        printf("$.each(plot.getXAxes(), function(_, axis) {\nvar opts =
axis.options;\nopts.panRange = true;\n});\n");
        printf("if  (!pan)  {  pan=true;  p.interactive=true;  plot  =
$.plot(\"#signal_%s_%s\",                                          [%s_%s],
options); }",name[i],param[i],name[i],param[i]);
        printf("} else\n");
            printf("if (this.value == 'zoomy') {\n");
        printf("s.mode=\"y\";\n");
        printf("if  (pan)  {  pan=false;  p.interactive=false;  plot  =
$.plot(\"#signal_%s_%s\",                                          [%s_%s],
options); }",name[i],param[i],name[i],param[i]);
            printf("}\n");
        printf("});\n");
        printf("});\n");
        printf("$(                                          \".signal_bar%s
button\" ).button();\n",correlation?"_square":"");
        printf("function autoscale( event ) {\n");
        printf("var axes = plot.getAxes(),");
        printf("xaxis = axes.xaxis.options,");
        printf("yaxis = axes.yaxis.options;\n");
            printf("    if    (event.target.value    ==    \"autox\"    ||
event.target.value  ==  \"autoxy\")  {  xaxis.min  =  null;  xaxis.max  =
null; }\n");
        printf("    if    (event.target.value    ==    \"autoy\"    ||
event.target.value  ==  \"autoxy\")  {  yaxis.min  =  null;  yaxis.max  =
null; }\n");
        printf("plot.setupGrid(); plot.draw();");
        printf("    }\n");
        printf("$( \"#autox_%s_%s\" ).click(autoscale);\n",name[i],param[
i]);
        printf("$( \"#autoy_%s_%s\" ).click(autoscale);\n",name[i],param[
i]);
        printf("$( \"#autoxy_%s_%s\" ).click(autoscale);\n",name[i],param
[i]);
```

```
        printf("});\n");
        i++;
        }

        printf("</script>\n");
        printf("</head>\n");
        printf("<body>\n");

        printf("<div class=\"main\">");
        printf("<div class=\"bar\"><h3 class=\"bar\">Istituto Nazionale
di Geofisica e Vulcanologia<br>Osservatorio Etneo</h1>");
        printf("<h2 class=\"bar\">M.te Etna Strainmeters monitoring
network</h2><h2 class=\"bar\">Remote Diagnostic and Administration
Tools</h2></div>");
        i=0;
        while(param[i][0]) {
        printf("<div
class=\"signal_frame%s\">",correlation?"_square":"");
        printf("<div id=\"signal_%s_%s\"
class=\"signal%s\"></div>",name[i],param[i],correlation?"_square":"");
        printf("<div
class=\"signal_bar%s\">\n",correlation?"_square":"");
        printf("<fieldset>\n");
        printf("<label class=\"signal_bar\" for=\"zoomx_%s_%s\">Zoom
X</label>\n",name[i],param[i]);
        printf("<input type=\"radio\" name=\"signal_bar_%s_%s\"
id=\"zoomx_%s_%s\"
value=\"zoomx\">\n",name[i],param[i],name[i],param[i]);
        printf("<label class=\"signal_bar\" for=\"zoomy_%s_%s\">Zoom
Y</label>\n",name[i],param[i]);
        printf("<input type=\"radio\" name=\"signal_bar_%s_%s\"
id=\"zoomy_%s_%s\" checked=\"checked\"
value=\"zoomy\">\n",name[i],param[i],name[i],param[i]);
        printf("<label class=\"signal_bar\" for=\"zoomxy_%s_%s\">Zoom
XY</label>\n",name[i],param[i]);
        printf("<input type=\"radio\" name=\"signal_bar_%s_%s\"
id=\"zoomxy_%s_%s\"
value=\"zoomxy\">\n",name[i],param[i],name[i],param[i]);
        printf("<label class=\"signal_bar\" for=\"panx_%s_%s\">Pan
X</label>\n",name[i],param[i]);
        printf("<input type=\"radio\" name=\"signal_bar_%s_%s\"
id=\"panx_%s_%s\" value=\"panx\">\n",name[i],param[i],name[i],param[i]);
        printf("<label class=\"signal_bar\" for=\"pany_%s_%s\">Pan
Y</label>\n",name[i],param[i]);
        printf("<input type=\"radio\" name=\"signal_bar_%s_%s\"
id=\"pany_%s_%s\" value=\"pany\">\n",name[i],param[i],name[i],param[i]);
        printf("<label class=\"signal_bar\" for=\"panxy_%s_%s\">Pan
XY</label>\n",name[i],param[i]);
        printf("<input type=\"radio\" name=\"signal_bar_%s_%s\"
id=\"panxy_%s_%s\"
value=\"panxy\">\n",name[i],param[i],name[i],param[i]);
        printf("<button type=\"button\" id=\"autox_%s_%s\"
class=\"signal_bar\" value=\"autox\">AutoScale
X</button>\n",name[i],param[i]);
        printf("<button type=\"button\" id=\"autoy_%s_%s\"
class=\"signal_bar\" value=\"autoy\">AutoScale
Y</button>\n",name[i],param[i]);
        printf("<button type=\"button\" id=\"autoxy_%s_%s\"
class=\"signal_bar\" value=\"autoxy\">AutoScale
XY</button>\n",name[i],param[i]);
        printf("</fieldset>\n");
        printf("</fieldset>\n");
```

```
                printf("</div>");
                printf("<div                                class=\"signal_title%s\"><h1
        class=\"signal_title\">Signal          <b>%s</b>          of          station
        <b>%s</b></h1></div>",correlation?"_square":"",param[i],name[i]);
                printf("</div>");
                i++;
                }
                printf("</div>");
                printf("</body></html>");
}

void showSyntesis(void)
{
        int i;
        char tclass[64]="black";
        char tclass_diagnosis[64]="black";
        char delay_message[32];

        DT today;

        getTime(today);

        printf("Content-type: text/plain\r\n\r\n");

                printf("Updated
        at %2.2hd/%2.2hd/%2.2hd %2.2hd:%2.2hd:%2.2hd\n",today.day,today.month,tod
        ay.year,today.hour,today.min,today.sec);
        checkDDNS(true);

        for (i=0;i<nstations;i++) {
                switch(stations[i].status) {
                        case GREEN:
                                strcpy(tclass_diagnosis,"green");
                                break;
                        case YELLOW:
                                strcpy(tclass_diagnosis,"yellow");
                                break;
                        case RED:
                                strcpy(tclass_diagnosis,"red");
                                break;
                }
                if ((stations[i].router  != -1 && stations[i].router)  ||
                    (stations[i].shoebox != -1 && stations[i].shoebox) ||
                    (stations[i].iboot != -1 && stations[i].iboot) ||
                    (stations[i].rabbit != -1 && stations[i].rabbit) ||
                                    (stations[i].ssh  !=  -1  &&  stations[i].ssh))
                strcpy(tclass_diagnosis,"yellow");                              else
                strcpy(tclass_diagnosis,"green");
                switch(stations[i].delay) {
                        case 0:
                                strcpy(tclass,"green");
                                break;
                        case 1:
                                strcpy(tclass,"yellow");
                                break;
                        default:
                                strcpy(tclass,"red");
                                strcpy(tclass_diagnosis,"red");
                                break;
                }

                        if (stations[i].delay) sprintf(delay_message,"late of %d
```

```
         days",stations[i].delay); else strcpy(delay_message,"on time");

                printf("%s\n%s\n%s\n%s\n%s\n%s\n%s\n",stations[i].name,delay_mess
         age,tclass,stations[i].diagnosis,stations[i].alert,tclass_diagnosis,stat
         ions[i].general_status);
        }
}

void showHome(void)
{
        int i;
        char tmp[512];
        char tclass[64]="station_black";
        char tclass_diagnosis[64]="station_black";
        char delay_message[32];
        char protocol[32];

        DT today;

        getTime(today);

                printf("Content-type: text/html\r\n\r\n");
                printf("<html>");
                printf("<head>");
                printf("<meta charset=\"UTF-8\">\n");
                printf("<link        rel=\"stylesheet\"        type=\"text/css\"
       href=\"/web.admin.css\">");
                printf("</head>");
                printf("<body>");

                printf("<script language=\"JavaScript\">");
                printf("function resetServer(name){\nvar r=confirm(\"Do you want
       reset  \"  +  name  +  \"  Server?\");\nif  (r)  window.location=\"/cgi-
       bin/web.admin.fcgi?source=web&what=reset&who=\"+name;\n}");

                printf("</script>");

                printf("<div class=\"main\">");
                printf("<div  class=\"bar\"><h3  class=\"bar\">Istituto  Nazionale
       di Geofisica e Vulcanologia<br>Osservatorio Etneo</h1>");
                printf("<h2  class=\"bar\">M.te  Etna  Strainmeters  monitoring
       network</h2><h2  class=\"bar\">Remote  Diagnostic  and  Administration
       Tools</h2></div>");
                printf("<div class=\"menu_bar\">");
                printf("<div                              class=\"dropdown\"><button
       class=\"dropbtn\">Diagnostic</button><div class=\"dropdown-content\">");
                printf("<h3     class=\"menu_bar\"><a     class=\"station_black\"
       href=\"/data\" target=\"_blank\">Downloaded data</a></h3>");
                printf("<h3     class=\"menu_bar\"><a     class=\"station_black\"
       href=\"/cgi-bin/web.admin.fcgi?servers=show\"    target=\"_blank\">Server
       Status</a></h3>");
                printf("<h3     class=\"menu_bar\"><a     class=\"station_black\"
       href=\"/cgi-bin/web.admin.fcgi?signals=pwr\"  target=\"_blank\">Batteries
       Voltage</a></h3>");
                printf("<h3     class=\"menu_bar\"><a     class=\"station_black\"
       href=\"/cgi-bin/web.admin.fcgi?signals=slope\"
       target=\"_blank\">Batteries Slope</a></h3>");
                printf("<h3     class=\"menu_bar\"><a     class=\"station_black\"
       href=\"/cgi-bin/web.admin.fcgi?signals=correlation\"
       target=\"_blank\">Batteries Correlation</a></h3>");
                printf("<h3     class=\"menu_bar\"><a     class=\"station_black\"
       href=\"/cgi-bin/web.admin.fcgi?signals=tides\"    target=\"_blank\">Tides
```

```
Analysis</a></h3>");
        printf("</div></div>");
        printf("<div                              class=\"dropdown\"><button
class=\"dropbtn\">Administration</button><div           class=\"dropdown-
content\">");
        printf("<h3     class=\"menu_bar\"><a     class=\"station_black\"
href=\"#\"      onclick=\"resetServer('ctstdata')\">Reset      Acquisition
Server</a></h3>");
        printf("<h3     class=\"menu_bar\"><a     class=\"station_black\"
href=\"#\"         onclick=\"resetServer('ctdyndns')\">Reset         DDNS
Server</a></h3>");
        printf("</div></div>");
        printf("<div                              class=\"dropdown\"><button
class=\"dropbtn\">RDAT</button><div class=\"dropdown-content\">");
        printf("<h3     class=\"menu_bar\"><a     class=\"station_black\"
href=\"/cgi-bin/web.admin.fcgi?inserts=show\"      target=\"_blank\">Show
Messages</a></h3>");
        printf("<h3     class=\"menu_bar\"><a     class=\"station_black\"
href=\"/rdat.apk\">Android Application</a></h3>");
        printf("</div></div></div>");
        checkDDNS();

        printf("<table class=\"stations\" align=\"center\">");
        printf("<tr                                 class=\"stations\"><th
class=\"stations\">Name</th><th       class=\"stations\">Download</th><th
class=\"stations\">WWW</th><th
class=\"stations\">Status/Diagnosis</th><th       class=\"stations\">Last
data</th><th class=\"stations\">Signals</th></tr>");
        for (i=0;i<nstations;i++) {
                switch(stations[i].status) {
                        case GREEN:
                                strcpy(tclass_diagnosis,"station_green");
                                break;
                        case YELLOW:
                                strcpy(tclass_diagnosis,"station_yellow");
                                break;
                        case RED:
                                strcpy(tclass_diagnosis,"station_red");
                                break;
                }
                if ((stations[i].router  != -1 && stations[i].router)  ||
                    (stations[i].shoebox != -1 && stations[i].shoebox) ||
                    (stations[i].iboot != -1 && stations[i].iboot) ||
                    (stations[i].rabbit != -1 && stations[i].rabbit) ||
                    (stations[i].ssh    !=    -1    &&    stations[i].ssh))
strcpy(tclass_diagnosis,"station_yellow");                              else
strcpy(tclass_diagnosis,"station_green");
                switch(stations[i].delay) {
                        case 0:
                                strcpy(tclass,"station_green");
                                break;
                        case 1:
                                strcpy(tclass,"station_yellow");
                                break;
                        default:
                                strcpy(tclass,"station_red");
                                strcpy(tclass_diagnosis,"station_red");
                                break;
                }
                tmp[0]=0;
                if (stations[i].ip[0]) {
                        if              (!strcmp(stations[i].name,"degi"))
```

```
        protocol[0]=0; else strcpy(protocol,"s");
                                sprintf(tmp,"<a            class=\"station_black\"
        target=\"_blank\"        href=\"http%s://%s:25328/\">Router</a>,        <a
        class=\"station_black\"                                  target=\"_blank\"
        href=\"http://%s:32454/data/\">Data</a>,     <a     class=\"station_black\"
        target=\"_blank\"                             href=\"http://%s:32454/cgi-
        bin/status.cgi/\">Status</a>,          <a          class=\"station_black\"
        target=\"_blank\"   href=\"http://%s:32454/%s/Plots.html\">Plots</a>,   <a
        class=\"station_black\"                                  target=\"_blank\"
        href=\"http://%s:54235/\">iBoot</a>",protocol,stations[i].ip,stations[i].
        ip,stations[i].ip,stations[i].ip,stations[i].name,stations[i].ip);
                        }
                        if (stations[i].delay) sprintf(delay_message,"late of %d
        days",stations[i].delay); else strcpy(delay_message,"on time");
                        printf("<tr>");
                        printf("<td                          class=\"stations\"><h1
        class=\"station_black\">%s</h1></td>\
                                <td                          class=\"stations\"><h1
        class=\"%s\">%s</h1></td>\
                                <td                          class=\"stations\"><h1
        class=\"station_black\">%s</h1></td>\
                                <td       class=\"stations\"><a       class=\"%s\"
        target=\"_blank\"                                    href=\"/cgi-
        bin/web.admin.fcgi?details=%s\">%s</a><br><h1
        class=\"station_red_blink\">%s</h1></td>\
                                <td                          class=\"stations\"><h1
        class=\"station_black\"><a                          class=\"station_black\"
        href=\"/data/%s/%4.4hd/%2.2hd\"
        target=\"_blank\">%2.2hd/%2.2hd/%4.4hd %2.2hd:%2.2hd:%2.2hd</a></h1></td>
        \
                                <td                          class=\"stations\"><h1
        class=\"station_black\"><a    class=\"station_black\"    target=\"_blank\"
        href=\"/cgi-bin/web.admin.fcgi?signals=%s\">view</a></h1></td>",\
                                stations[i].name,
                                tclass,delay_message,
                                tmp,
                                tclass_diagnosis,
                                stations[i].name,
                                stations[i].diagnosis,stations[i].alert,

         stations[i].name,stations[i].delay_dt.year,stations[i].delay_dt.month,

         stations[i].last_data.day,stations[i].last_data.month,stations[i].last_d
        ata.year,stations[i].last_data.hour,stations[i].last_data.min,stations[i]
        .last_data.sec,
                                stations[i].name);
                                printf("</tr>");
                }
                printf("</table>");
                printf("</div>");
                printf("</body></html>");
}

void showDetails(char *name)
{
        int i;
        char tmp[512];
        char tclass[64]="station_black";
        char sclass[64]="station_black";
        char iclass[64]="station_black";
        char tclass_diagnosis[64]="station_black";
        char ip[256];
```

```
char signal[256];
char protocol[8];
char delay_message[32];
int ipc=0;
int sc=0;

DT today;

getTime(today);

printf("Content-type: text/html\r\n\r\n");
printf("<html>");
printf("<head>");
printf("<meta charset=\"UTF-8\">\n");
        printf("<link        rel=\"stylesheet\"        type=\"text/css\"
href=\"/web.admin.css\">");
printf("</head>");
printf("<body>");

printf("<div class=\"main\">");
        printf("<div  class=\"bar\"><h3  class=\"bar\">Istituto  Nazionale
di Geofisica e Vulcanologia<br>Osservatorio Etneo</h1>");
        printf("<h2  class=\"bar\">M.te  Etna  Strainmeters  monitoring
network</h2><h2  class=\"bar\">Remote  Diagnostic  and  Administration
Tools</h2></div>");
printf("<br>");
printf("<table class\"details\">");

for (i=0;i<nstations;i++) if (!strcmp(stations[i].name,name)) break;
if (i == nstations) return;
switch(stations[i].status) {
                case GREEN:
                        strcpy(tclass_diagnosis,"station_green");
                        break;
                case YELLOW:
                        strcpy(tclass_diagnosis,"station_yellow");
                        break;
                case RED:
                        strcpy(tclass_diagnosis,"station_red");
                        break;
        }
            if ((stations[i].router  != -1 && stations[i].router)  ||
            (stations[i].shoebox != -1 && stations[i].shoebox) ||
            (stations[i].iboot != -1 && stations[i].iboot) ||
            (stations[i].rabbit != -1 && stations[i].rabbit) ||
                            (stations[i].ssh  !=  -1  &&  stations[i].ssh))
        strcpy(tclass_diagnosis,"station_yellow");                    else
        strcpy(tclass_diagnosis,"station_green");
        switch(stations[i].delay) {
                case 0:
                        strcpy(tclass,"station_green");
                        break;
                case 1:
                        strcpy(tclass,"station_yellow");
                        break;
                default:
                        strcpy(tclass,"station_red");
                        strcpy(tclass_diagnosis,"station_red");
                        break;
        }
        tmp[0]=0;
        ip[0]=0;
```

```
        signal[0]=0;
        if (stations[i].ip[0]) {
                        if              (!strcmp(stations[i].name,"degi"))
protocol[0]=0; else strcpy(protocol,"s");

 ipc=calculateHours(stations[i].ip_timestamp,today);
                        if          (ipc          >=          MAX_RESET_HOURS)
strcpy(iclass,"station_red"); else strcpy(iclass,"station_green");
                        sprintf(tmp,"<a             class=\"station_black\"
target=\"_blank\"               href=\"http%s://%s:25328/\">Router</a>,<a
class=\"station_black\"                                    target=\"_blank\"
href=\"http://%s:32454/data/\">Data</a>,<a        class=\"station_black\"
target=\"_blank\"                                href=\"http://%s:32454/cgi-
bin/status.cgi/\">Status</a>,<a class=\"station_black\" target=\"_blank\"
href=\"http://%s:32454/%s/Plots.html\">Plots</a>,<a
class=\"station_black\"                                    target=\"_blank\"
href=\"http://%s:54235/\">iBoot</a>",protocol,stations[i].ip,stations[i].
ip,stations[i].ip,stations[i].ip,stations[i].name,stations[i].ip);
                        sprintf(ip,"the  station  acquired  ip  number  %s
at   %2.2d/%2.2d/%4.4d   %2.2d:%2.2d:%2.2d   and   DDNS   processed   SMS
at %2.2d/%2.2d/%4.4d %2.2d:%2.2d:%2.2d (%d hours ago)",

 stations[i].ip,stations[i].ip_timestamp.day,stations[i].ip_timestamp.mon
th,stations[i].ip_timestamp.year,stations[i].ip_timestamp.hour,stations[i
].ip_timestamp.min,stations[i].ip_timestamp.sec,

 stations[i].ip_dt.day,stations[i].ip_dt.month,stations[i].ip_dt.year,sta
tions[i].ip_dt.hour,stations[i].ip_dt.min,stations[i].ip_dt.sec,ipc);
                }
                if (stations[i].signal[0]) {

 sc=calculateHours(stations[i].signal_timestamp,today);
                        if          (sc          >=          MAX_RESET_HOURS)
strcpy(sclass,"station_red"); else strcpy(sclass,"station_green");
                        sprintf(signal,"the     device     was     powered
at  %2.2d/%2.2d/%4.4d  %2.2d:%2.2d:%2.2d  with  signal  of  %s  (%d  hours
ago)",stations[i].signal_timestamp.day,stations[i].signal_timestamp.month
,stations[i].signal_timestamp.year,stations[i].signal_timestamp.hour,stat
ions[i].signal_timestamp.min,stations[i].signal_timestamp.sec,stations[i]
.signal,sc);
                }

                if  (stations[i].delay)  sprintf(delay_message,"late  of  %d
days",stations[i].delay); else strcpy(delay_message,"on time");

                printf("<tr><td                        class=\"details\"><h1
class=\"details\">Name:        </h1></td><td        class=\"details\"><h1
class=\"station_black\">%s</h1></td></tr>",stations[i].name);
                printf("<tr><td                        class=\"details\"><h1
class=\"details\">Download  status:  </h1></td><td  class=\"details\"><h1
class=\"%s\">%s</h1></td></tr>",tclass,delay_message);
                printf("<tr><td                        class=\"details\"><h1
class=\"details\">Web    links:    </h1></td><td    class=\"details\"><h1
class=\"station_black\">%s</h1></td></tr>",tmp);
                printf("<tr><td                        class=\"details\"><h1
class=\"details\">Last  Router  test:  </h1></td><td  class=\"details\"><h1
class=\"%s\">%s</h1></td></tr>",(stations[i].router     ||     ipc     >=
MAX_RESET_HOURS)?"station_red":"station_green",ip[0]?(stations[i].router?
FAIL_TEXT:OK_TEXT):"");
            printf("<tr><td class=\"details\"><h1 class=\"details\">Last
Shoebox              test:</h1></td><td              class=\"details\"><h1
class=\"%s\">%s</h1></td></tr>",(stations[i].shoebox     ||     ipc     >=
```

```
        MAX_RESET_HOURS)?"station_red":"station_green",ip[0]?(stations[i].shoebox
        ?FAIL_TEXT:OK_TEXT):"");
                    printf("<tr><td class=\"details\"><h1 class=\"details\">Last
        iBoot          test:          </h1></td><td          class=\"details\"><h1
        class=\"%s\">%s</h1></td></tr>",(stations[i].iboot      ||      ipc      >=
        MAX_RESET_HOURS)?"station_red":"station_green",ip[0]?(stations[i].iboot?F
        AIL_TEXT:OK_TEXT):"");
                    printf("<tr><td class=\"details\"><h1 class=\"details\">Last
        Rabbit          test:          </h1></td><td          class=\"details\"><h1
        class=\"%s\">%s</h1></td></tr>",(stations[i].rabbit||        ipc        >=
        MAX_RESET_HOURS)?"station_red":"station_green",ip[0]?(stations[i].rabbit?
        FAIL_TEXT:OK_TEXT):"");
                    printf("<tr><td class=\"details\"><h1 class=\"details\">Last
        SSH           test:          </h1></td><td          class=\"details\"><h1
        class=\"%s\">%s</h1></td></tr>",(stations[i].ssh      ||      ipc      >=
        MAX_RESET_HOURS)?"station_red":"station_green",ip[0]?(stations[i].ssh?FAI
        L_TEXT:OK_TEXT):"");
                    printf("<tr><td                          class=\"details\"><h1
        class=\"details\">Internet          Connection:          </h1></td><td
        class=\"details\"><h1 class=\"%s\">%s</h1></td></tr>",iclass,ip);
                    printf("<tr><td                          class=\"details\"><h1
        class=\"details\">Modem/Router:    </h1></td><td    class=\"details\"><h1
        class=\"%s\">%s</h1></td></tr>",sclass,signal);
                printf("<tr><td   class=\"details\"><h1   class=\"details\">Last
        data:   </h1></td><td   class=\"details\"><h1   class=\"station_black\"><a
        class=\"station_black\"                    href=\"/data/%s/%4.4hd/%2.2hd\"
        target=\"_blank\">%2.2hd/%2.2hd/%4.4hd</a></h1></td></tr>",stations[i].na
        me,stations[i].delay_dt.year,
             stations[i].delay_dt.month,stations[i].delay_dt.day,stations[i].dela
        y_dt.month,stations[i].delay_dt.year);
                    printf("<tr><td                          class=\"details\"><h1
        class=\"details\">Status/Diagnosis:  </h1></td><td  class=\"details\"><h1
        class=\"%s\">%s</h1><br><h1
        class=\"station_red_blink\">%s</h1></td></tr>",tclass_diagnosis,stations[
        i].diagnosis,stations[i].alert);
                 printf("</table>");
                printf("</div>");
                printf("</body></html>");
}

void showServers(void)
{
        int i;

                printf("Content-type: text/html\r\n\r\n");
                printf("<html>");
                printf("<head>");
                printf("<meta charset=\"UTF-8\">\n");
                printf("<link        rel=\"stylesheet\"        type=\"text/css\"
        href=\"/web.admin.css\">");
                printf("</head>");
                printf("<body>");
                printf("<div class=\"main\">");
                printf("<div  class=\"bar\"><h1  class=\"\">Istituto  Nazionale  di
        Geofisica e Vulcanologia<br>Osservatorio Etneo</h1>");
                printf("<h2   class=\"bar\">M.te   Etna   Strainmeters   monitoring
        network</h2><h2   class=\"bar\">Remote   Diagnostic   and   Administration
        Tools</h2></div>");
                printf("<table class=\"inserts\">");
                printf("<tr                                  class=\"inserts\"><th
        class=\"inserts\">Server</th><th class=\"inserts\">Status</th></tr>");
                for (i=0;i<nservers;i++) {
```

```
                        printf("<tr>");
                        printf("<td                    class=\"inserts\">%s</td><td
        class=\"inserts\">%s</td>",servers[i].name,servers[i].reset?"reset":"idle
        ");
                        printf("</tr>");
                }
                printf("</table>");
                printf("</div>");
                printf("</body></html>");
}

void showInserts(void)
{
        int i;

        printf("Content-type: text/html\r\n\r\n");
        printf("<html>");
                printf("<head>");
                printf("<meta charset=\"UTF-8\">\n");
                printf("<link        rel=\"stylesheet\"        type=\"text/css\"
        href=\"/web.admin.css\">");
                printf("</head>");
                printf("<body>");
                printf("<div class=\"main\">");
                printf("<div  class=\"bar\"><h1  class=\"\">Istituto  Nazionale  di
        Geofisica e Vulcanologia<br>Osservatorio Etneo</h1>");
                printf("<h2   class=\"bar\">M.te   Etna   Strainmeters   monitoring
        network</h2><h2   class=\"bar\">Remote   Diagnostic   and   Administration
        Tools</h2></div>");
                printf("<table class=\"inserts\">");
                printf("<tr  class=\"inserts\"><th  class=\"inserts\">Date</th><th
        class=\"inserts\">Time</th><th           class=\"inserts\">Source</th><th
        class=\"inserts\">What</th><th class=\"inserts\">Who</th></tr>");
                int                    min=ninserts<MAX_SHOWED_INSERTS?0:ninserts-
        MAX_SHOWED_INSERTS;
                for (i=ninserts-1;i>=min;i--) {
                        printf("<tr>");
                        printf("<td                    class=\"inserts\">%s</td><td
        class=\"inserts\">%s</td><td                    class=\"inserts\">%s</td><td
        class=\"inserts\">%s</td><td
        class=\"inserts\">%s</td>",inserts[i].date,inserts[i].time,inserts[i].sou
        rce,inserts[i].what,inserts[i].who);
                        printf("</tr>");
                }
                printf("</table>");
                printf("</div>");
                printf("</body></html>");
}

void showError(const char *err)
{
        printf("Content-type: text/html\r\n\r\n");
        printf("<html><body>Error: %s</body></html>",err);
}

void redirectToHome(void)
{
                printf("Content-type: text/html\r\n\r\n");
                printf("<html><script   language=\"JavaScript\">var   url=\"/cgi-
        bin/web.admin.fcgi\"\nif (document.images)\n");
                        printf("location.replace(url);\nelse\nlocation.href        =
        url;\n</script>\n");
```

```
                printf("<a                    href=\"/cgi-bin/web.admin.fcgi\">Wait
        please...</a>\n</html>");
}

void start(void)
{
        int n=0;
        char *s;

        SaveLog();

        s=getenv("REQUEST_URI");

        if   (strlen(s)   >=   MAX_QUERY_SIZE)   {   showError("Query   overflow");
return; }

        if (!getParameters(s,parameters,n,MAX_PARAMETERS)) {
                                            if (!n) { showHome(); return; }
                                                    if   (n   ==   MAX_PARAMETERS)
        showError("Too many parameters");

        else

        showError("Wrong query");
                                                          return;
                }

                if (n == 1) {
                        if (!strcmp(parameters[0].name,RESET_TEXT)) {
                                printf("Content-type: text/html\r\n\r\n");
                                if                    (getReset(parameters[0].value))
        printf("true"); else printf("false");
                                return;
                        }
                        if (!strcmp(parameters[0].name,INSERTS_TEXT)) {
                                showInserts();
                                return;
                        }
                        if (!strcmp(parameters[0].name,SERVERS_TEXT)) {
                                showServers();
                                return;
                        }
                    if (!strcmp(parameters[0].name,DETAILS_TEXT)) {
                                showDetails(parameters[0].value);
                                return;
                        }
                if (!strcmp(parameters[0].name,SIGNALS_TEXT)) {
                                showSignals(parameters[0].value);
                                return;
                        }

                if (!strcmp(parameters[0].name,SYNTESIS_TEXT)) {
                                showSyntesis();
                                return;
                        }
                 }

                if (!insert(parameters,n)) { showError("Wrong insert"); return; }

                redirectToHome();
}
```

```
int main ()
{
    int i;

    SaveLog("Main");
    #ifndef SIMPLE_TEST
    memset(&servers,0,sizeof(servers));
    memset(&stations,0,sizeof(stations));
    for (i=0;i<MAX_STATIONS;i++) {
        stations[i].router=-1;
        stations[i].iboot=-1;
        stations[i].ssh=-1;
        stations[i].shoebox=-1;
        stations[i].rabbit=-1;
    }
    SaveLog("Load Inserts");
    loadInserts();
    #endif
    SaveLog("Start IDLE");
    while (FCGI_Accept() >= 0) {
        #ifdef  SIMPLE_TEST
                printf("Content-type: text/html\r\n\r\n");
                printf("ok");
        #else
                start();
        #endif
    } /* while */

    return 0;
     }
```

Appendice A3. Codice sorgente: Makefile web.admin

```
# fcgi web.admin makefile

CC=g++
OPTIONS= -c
ApacheServer=/etc/init.d/apache2

all: web.admin.o test.o
	$(ApacheServer) stop
	$(CC) web.admin.o -o web.admin -Xlinker -lm -lfcgi
	cp web.admin ../cgi-bin/web.admin.fcgi
	chown rdat:rdat ../cgi-bin/web.admin.fcgi
	#chmod u+s ../cgi-bin/web.admin.fcgi
	#chmod g+s bin/web.admin.fcgi
	$(CC) test.o -o test
	cp test ../cgi-bin/test
	chown rdat:rdat ../cgi-bin/test
	$(ApacheServer) start
	rm web.admin
	-mkdir /var/log/web.admin
	-chmod 777 /var/log/web.admin

test.o: test.c
	$(CC) test.c $(OPTIONS)

web.admin.o: web.admin.cpp web.admin.h
	$(CC) web.admin.cpp $(OPTIONS)

clean:
	rm *.o
	rm web.admin
	rm test
```

Appendice A4. Codice sorgente: web.admin.css

```
html, body {
  height: 100%;
}

body {
  margin: 0;
  padding: 0;
  text-align: center;
  background-color: #FFFFFF;
}

div.bar {
  background-image: url(images/bar.jpg);
  position: relative;
  width: 1280px;
  margin: 0 auto;
  padding: 0;
  height: 193px;
  top: 0px;
}

div.main {
  position: relative;
  width: 1280px;
  margin: 0 auto;
  padding: 0;
  top: 0px;
  background-color: #ECECEC;
  display:block;
  overflow:auto;
}

div.submain {
  position: relative;
  width: 1024px;
  margin: 0 auto;
  padding: 0;
  top: 0px;
  background-color: #ECECEC;
  display:block;
  overflow:auto;
  text-align:left;
}

label.signal_bar {
        display: block;
        width: 96px;
}

button.signal_bar {
        display: block;
        width: 130px;
}

div.signal_frame {
        box-sizing: border-box;
        width: 1200px;
        height: 390px;
        padding: 30px 15px 30px 15px;
        margin: 15px auto 20px auto;
```

```
        border: 1px solid #ddd;
        background: #fff;
        background: linear-gradient(#f6f6f6 0, #fff 50px);
        background: -o-linear-gradient(#f6f6f6 0, #fff 50px);
        background: -ms-linear-gradient(#f6f6f6 0, #fff 50px);
        background: -moz-linear-gradient(#f6f6f6 0, #fff 50px);
        background: -webkit-linear-gradient(#f6f6f6 0, #fff 50px);
        box-shadow: 0 3px 10px rgba(0,0,0,0.15);
        -o-box-shadow: 0 3px 10px rgba(0,0,0,0.1);
        -ms-box-shadow: 0 3px 10px rgba(0,0,0,0.1);
        -moz-box-shadow: 0 3px 10px rgba(0,0,0,0.1);
        -webkit-box-shadow: 0 3px 10px rgba(0,0,0,0.1);

  display:block;
  overflow:hidden;
  text-align:left;
}

div.signal_frame_square {
        box-sizing: border-box;
        width: 880px;
        height: 650px;
        padding: 30px 15px 30px 15px;
        margin: 15px auto 20px auto;
        border: 1px solid #ddd;
        background: #fff;
        background: linear-gradient(#f6f6f6 0, #fff 50px);
        background: -o-linear-gradient(#f6f6f6 0, #fff 50px);
        background: -ms-linear-gradient(#f6f6f6 0, #fff 50px);
        background: -moz-linear-gradient(#f6f6f6 0, #fff 50px);
        background: -webkit-linear-gradient(#f6f6f6 0, #fff 50px);
        box-shadow: 0 3px 10px rgba(0,0,0,0.15);
        -o-box-shadow: 0 3px 10px rgba(0,0,0,0.1);
        -ms-box-shadow: 0 3px 10px rgba(0,0,0,0.1);
        -moz-box-shadow: 0 3px 10px rgba(0,0,0,0.1);
        -webkit-box-shadow: 0 3px 10px rgba(0,0,0,0.1);

  display:block;
  overflow:hidden;
  text-align:left;
}

div.signal_square {
        width: 700px;
        height: 600px;
        font-size: 14px;
        line-height: 1.2em;
}

div.signal_title_square {
  position: relative;
  width: 390px;
  margin: 0 auto;
  padding: 0;
  height: 20px;
  top: -1025px;
  left: -220px;
}

div.signal_bar_square {
  position: relative;
  width: 160px;
```

```
  margin: 0 auto;
  padding: 0;
  height: 400px;
  top: -593px;
  left: 353px;
}

div.signal {
        width: 1060px;
        height: 353px;
        font-size: 14px;
        line-height: 1.2em;
}

div.signal_title {
  position: relative;
  width: 350px;
  margin: 0 auto;
  padding: 0;
  height: 20px;
  top: -780px;
  left: -420px;
}

h1.signal_title {
    margin-top: 0px;
    margin-left: 0px;
    margin-right: 0px;
    font-family: Garamond, serif;
    font-style: italic;
    font-size: 16pt;
    font-weight: normal;
    color: #000000;
}

div.signal_bar {
  position: relative;
  width: 160px;
  margin: 0 auto;
  padding: 0;
  height: 400px;
  top: -346px;
  left: 518px;
}

table.stations {
    border-collapse: collapse;
    width: 1280px;
}

table.stations, td.stations, th.stations {
    border: 1px solid black;
}

table.details {
    border-collapse: collapse;
    width: 100%;
}

table.details, td.details, th.details {
    border: 0px solid black;
}
```

```
td.stations {
    text-align:center;
    vertical-align:middle;
}

a.station_green {
        color: #009933;
}

h1.station_green {
    margin-top: 0px;
    margin-left: 0px;
    margin-right: 0px;
    font-family: Garamond, serif;
    font-style: italic;
    font-size: 16pt;
    font-weight: normal;
    color: #009933;
}

a.station_red {
        color: #ff0000;
}

h1.station_red {
    margin-top: 0px;
    margin-left: 0px;
    margin-right: 0px;
    font-family: Garamond, serif;
    font-style: italic;
    font-size: 16pt;
    font-weight: normal;
    color: #ff0000;
}

h1.station_red_blink {
    margin-top: 0px;
    margin-left: 0px;
    margin-right: 0px;
    font-family: Garamond, serif;
    font-style: italic;
    font-size: 16pt;
    font-weight: normal;
    color: #ff0000;
    animation: blink 1s steps(5, start) infinite;
    -webkit-animation: blink-animation 1s steps(5, start) infinite;
}

@keyframes blink {
  to {
    visibility: hidden;
  }
}

@-webkit-keyframes blink {
  to {
    visibility: hidden;
  }
}

a.station_yellow {
```

```
        color: #e6b800;
}

h1.station_yellow {
    margin-top: 0px;
    margin-left: 0px;
    margin-right: 0px;
    font-family: Garamond, serif;
    font-style: italic;
    font-size: 16pt;
    font-weight: normal;
    color: #e6b800;
}

h1.station_black {
    margin-top: 0px;
    margin-left: 0px;
    margin-right: 0px;
    font-family: Garamond, serif;
    font-style: italic;
    font-size: 16pt;
    font-weight: normal;
    color: #000000;
}

h1.details {
    margin-top: 0px;
    margin-left: 0px;
    margin-right: 0px;
    font-family: Garamond, serif;
    font-style: italic;
    font-size: 14pt;
    font-weight: bold;
    color: #000000;
}

a.station_black {
        color: #000000;
}

table.inserts {
    border-collapse: collapse;
    width: 100%;
}

table.inserts, td.inserts, th.inserts {
    border: 1px solid black;
}

td.inserts {
    text-align:center;
    vertical-align:middle;
}

h1.network {
    display: block;
    font-size: 3em;
    margin-top: 0.67em;
    margin-bottom: 0.67em;
    margin-left: 0;
    margin-right: 0;
    font-weight: bold;
```

```
    text-align: center;
}

h1.functions {
    display: block;
    font-size: 2em;
    margin-top: 0.67em;
    margin-bottom: 0.67em;
    margin-left: 0;
    margin-right: 0;
    font-weight: bold;
}

h3.bar {
    margin-top: 0px;
    margin-left: 0px;
    margin-right: 0px;
    font-family: Garamond, serif;
    font-style: italic;
    color: rgb(0, 0, 0);
    font-size: 24pt;
    font-weight: bold;
        }

h2.bar {
    margin-top: 0px;
    margin-left: 0px;
    margin-right: 0px;
    font-family: Garamond, serif;
    font-style: italic;
    color: rgb(0, 0, 0);
    font-size: 16pt;
    font-weight: bold;
        }

h1 {
    display: inline;
    line-height: 20pt;
}

.dropbtn {
    background-color: transparent;
    color: black;
    padding: 5px;
    font-size: 16px;
    border: none;
    cursor: pointer;
}

.dropdown {
    position: relative;
    left: -500px;
    display: inline-block;
}

.dropdown-content {
    display: none;
    position: absolute;
    background-color: #f9f9f9;
    min-width: 230px;
    box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
    z-index: 1;
```

```
}

.dropdown-content a {
    color: black;
    padding: 12px 16px;
    text-decoration: none;
    display: block;
}

.dropdown-content a:hover {background-color: #f1f1f1}

.dropdown:hover .dropdown-content {
    display: block;
}

.dropdown:hover .dropbtn {
    background-color: #f9f9f9;
}

div.menu_bar {
        padding:5px 0px;
        background: -moz-linear-gradient(top, #f9f9f9 0%, #c0c0c0 100%); /*
    firefox */
            box-shadow: 0 1px 2px #fff, 0 -1px 1px #666, inset 0 -1px 1px
    rgba(0,0,0,0.5), inset 0 1px 1px rgba(255,255,255,0.8);
            -moz-box-shadow: 0 1px 2px #fff, 0 -1px 1px #666, inset 0 -1px 1px
    rgba(0,0,0,0.5), inset 0 1px 1px rgba(255,255,255,0.8);
            -webkit-box-shadow: 0 1px 2px #fff, 0 -1px 1px #666, inset 0 -1px
    1px rgba(0,0,0,0.5), inset 0 1px 1px rgba(255,255,255,0.8);
            background: -webkit-gradient(linear, left top, left bottom, color-
    stop(0%,#f9f9f9), color-stop(100%,#c0c0c0)); /* webkit */font-size:20px
}

h3.menu_bar {
    text-align: left;
    padding: 0px;
    margin-top: 0px;
    margin-left: 0px;
    margin-right: 0px;
    font-family: Garamond, serif;
    font-style: italic;
    color: rgb(0, 0, 0);
    font-size: 12pt;
    font-weight: normal;
    display:inline
        }
```

Appendice A5. Codice sorgente: delay.h

```
#ifndef _DELAY_H
#define _DELAY_H

#define NSTATIONS     4
#define MAXIMUM_DELAY 2

#endif
```

Appendice A6. Codice sorgente: delay.cpp

```
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <stdlib.h>
#include "delay.h"
#include <string.h>

const char *stations[NSTATIONS]={"degi","druv","dpdn","dmsc"};
int delays[NSTATIONS];

int GetDays(int mon,int year)
{
        int days[12]={31,0,31,30,31,30,31,31,30,31,30,31};
        int v;

        v=days[mon-1];
        if (v == 0) {
                    if (((year/4)*4) != year) { v=28; }
                                              else
                                              { v=29; }
                    }
        return(v);
}

char cmd[1024];

int main(int argc,char *argv[])
{
        char filename[128];

        int i;
        int year,month,day;

        time_t t = time(NULL);
        struct tm tm = *localtime(&t);

        i=0;
        for (i=0;i<NSTATIONS;i++) {
                year=tm.tm_year+1900;
                day=tm.tm_mday;
                month=tm.tm_mon+1;
                delays[i]=0;
                while(true) {

        sprintf(filename,"/data/%s/%4.4d/%2.2d/%s.s1a.%4.4d.%2.2d.%2.2d.00.00.00
       .1s.sac",

        stations[i],year,month,stations[i],year,month,day);
                        if (access(filename,F_OK) != -1) {
                                sprintf(cmd,"wget                                -qO-
         \"http://user:password@magftp.ct.ingv.it:2016/cgi-
         bin/web.admin.fcgi?source=ctstdata&what=download_delay&who=%s:%d:%4.4hd%
         2.2hd%2.2hd\" > /dev/null 2>&1",stations[i],delays[i],year,month,day);
                                system(cmd);
                        break;
                        }
                        day--;
                        if (!day) {
                                month--;
```

```
                        if (!month) { year--; month=12; }
                        day=GetDays(month,year);
                    }
                    delays[i]++;
                }
        }

        bool reboot=true;
        for (i=0;i<NSTATIONS;i++) {
                    printf("%s %d\n",stations[i],delays[i]);
                    reboot &= delays[i] >= MAXIMUM_DELAY;
        }

        if (reboot && argc == 2 && !strcmp(argv[1],"reboot")) {
                    sprintf(cmd,"wget                                    -qO-
        \"http://user:password@magftp.ct.ingv.it:2016/cgi-
        bin/web.admin.fcgi?source=delay&what=reboot&who=ctstdata\"   >   /dev/null
        2>&1");
                    system(cmd);
                system("reboot");
        }

        return 0;
}
```

Appendice A7. Codice sorgente: rdat.reset.h

```
#ifndef  _RDAT_RESET_H
#define  _RDAT_RESET_H

#ifdef _ENABLE_DAEMON
#define HOME_DIRECTORY "/var/rdat.reset"
#define PID_FILE "/var/run/rdat.reset.pid"
#else
#define HOME_DIRECTORY "/home/strain/Shoebox/rdat.reset"
#define _DISABLE_DELETE
#endif

#define RDAT_QUERY_DELAY_TIME 10

#define MAX_PAGE_SIZE 50000

#define        RDAT_URL        "http://user:password@magftp.ct.ingv.it:2016/cgi-
bin/web.admin.fcgi?reset=ctstdata"

#endif //_RDAT_RESET_H
```

Appendice A8. Codice sorgente: rdat.reset.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>
#include <time.h>
#include <fcntl.h>
#include <linux/fs.h>
#include <linux/limits.h>
#include <sys/io.h>
#include <stdio.h>
#include <curl/curl.h>
#include "rdat.reset.h"

void error(const char *msg) { perror(msg); exit(0); }

FILE *HLOG=stdout;

void trim(char *line)
{
        int j=0,i;

        if (!line[0]) return;

        i=0;
        while(line[i]) {
                                if (line[i] == '\t') line[i]=0x20;
                                i++;
        }

        i=0;
        while(line[i] && (line[i] <= 0x20 || line[i] >= 0x80)) i++;
        //while(line[i] == 0x20 || line[i] == 0x0a || line[i] == 0x09) i++;
        if (!line[i]) { line[0]=0; return; }

        while(line[j+i]) {
                line[j]=line[j+i];
                j++;
        }

        line[j]=0;

        while(line[j] <= 0x20 || line[j] >= 0x80) j--;
        //while(line[j]  ==  0x20  ||  line[j]  ==  0x0a  ||  line[j]  ==  0x09
|| !line[j]) j--;
        line[j+1]=0;
}

void writeDate(void)
{
            time_t t = time(NULL);
            struct tm tm = *localtime(&t);
            printf("%4.4hd-%2.2hd-%2.2hd  %2.2hd:%2.2hd:%2.2hd  ",tm.tm_year  +
    1900,tm.tm_mon + 1,tm.tm_mday,tm.tm_hour,tm.tm_min,tm.tm_sec);
}

#ifdef _ENABLE_DAEMON
void daemon(void)
{
```

```
    pid_t pid;

    pid = fork ();
    if (pid == -1) exit(EXIT_FAILURE);
        else if (pid != 0)
            exit (EXIT_SUCCESS);

    if (setsid ( ) == -1) exit(EXIT_FAILURE);

    if (chdir (HOME_DIRECTORY) == -1) return exit(EXIT_FAILURE);

    close(STDIN_FILENO);
    close(STDOUT_FILENO);
    close(STDERR_FILENO);

    char filename[256];
    sprintf(filename,"%s/rdat.reset.run.log",HOME_DIRECTORY);
    open("/dev/null",O_RDWR);
    open(filename,O_RDWR);
    open(filename,O_RDWR);
    HLOG=fopen(filename,"at+");

    FILE *handle=fopen(PID_FILE,"wt+");
    fprintf(handle,"%d\n",getpid());
    fclose(handle);

    setvbuf(stdout, NULL, _IONBF, 0);
}
#endif

char page[MAX_PAGE_SIZE];
int ipage;

size_t writeFunction(char *ptr, size_t size, size_t nmemb, void *userdata)
{
    size_t n = size * nmemb;

    if   (ipage+n   >=   MAX_PAGE_SIZE)     {   writeDate();   printf("RDAT   buffer
overflow\n"); return 0; }

    memcpy(page+ipage, ptr, n);
    ipage += n;
    page[ipage]=0;
    return n;
}

int main(int argc,char *argv[])
{
    CURL *curl=NULL;
    CURLcode res;

     #ifdef _ENABLE_DAEMON
        daemon();
    #else
        setvbuf(stdout, NULL, _IONBF, 0);
    #endif

    writeDate();
    printf("RDAT reset start\n");

    while(true) {
                writeDate();
```

```
            printf("RDAT reset processing\n");

            if (!curl) {
                    curl = curl_easy_init();
                    if (curl) {
                        curl_easy_setopt(curl, CURLOPT_URL,RDAT_URL);
                        curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1L);
                        curl_easy_setopt(curl,             CURLOPT_WRITEFUNCTION,
writeFunction);
                    }
            }

            if (curl) {
                ipage=0;
                page[0]=0;
                res = curl_easy_perform(curl);

                if(res != CURLE_OK) {
                    writeDate();
                    printf("RDAT reset get value error");
                }
                else
                {
                    writeDate();
                    printf("[%s]\n",page);
                    if (!strcmp(page,"true")) system("reboot");
                }
            }

            writeDate();
            printf("sleep\n");
            sleep(RDAT_QUERY_DELAY_TIME);
        }

        curl_easy_cleanup(curl);
    #ifdef _ENABLE_DAEMON
    unlink(PID_FILE);
    fclose(HLOG);
    #endif
        return 0;
}
```

Appendice A9. Codice sorgente: rdat.reset.sh

```
#!/bin/sh
# kFreeBSD do not accept scripts as interpreters, using #!/bin/sh and sourcing.
if [ true != "$INIT_D_SCRIPT_SOURCED" ] ; then
    set "$0" "$@"; INIT_D_SCRIPT_SOURCED=true . /lib/init/init-d-script
fi
### BEGIN INIT INFO
# Provides:          skeleton
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Example initscript
# Description:       This file should be used to construct scripts to be
#                    placed in /etc/init.d.  This example start a
#                    single forking daemon capable of writing a pid
#                    file.  To get other behavoirs, implemend
#                    do_start(), do_stop() or other functions to
#                    override the defaults in /lib/init/init-d-script.
### END INIT INFO

# Author: Foo Bar <foobar@baz.org>
#
# Please remove the "Author" lines above and replace them
# with your own name if you copy and modify this script.

DESC="Remote Diagnostic and Administration Tool - Reset"
DAEMON=/usr/sbin/rdat.reset
```

Appendice A10. Codice sorgente: Makefile rdat.reset

```
all:    rdat.reset

rdat.reset: rdat.reset.cpp
        gcc -D_ENABLE_DAEMON -o rdat.reset rdat.reset.cpp -lcurl
        -killall rdat.reset
        -/etc/init.d/rdat.reset stop
        -mkdir /var/rdat.reset
        chmod 777 /var/rdat.reset
        cp -f rdat.reset.sh /etc/init.d/rdat.reset
        cp -f rdat.reset /usr/sbin/rdat.reset
        -ln -s ../init.d/rdat.reset /etc/rc2.d/S99rdat.reset
        /etc/init.d/rdat.reset start
        ps -aux | grep rdat.reset

clean:
        rm -f rdat.reset
        rm -f rdat.reset.o
```

ISSN 1590-2595

Quaderni di Geofisica

http://istituto.ingv.it/l-ingv/produzione-scientifica/quaderni-di-geofisica/

I Quaderni di Geofisica coprono tutti i campi disciplinari sviluppati all'interno dell'INGV, dando particolare risalto alla pubblicazione di dati, misure, osservazioni e loro elaborazioni anche preliminari, che per tipologia e dettaglio necessitano di una rapida diffusione nella comunità scientifica nazionale ed internazionale. La pubblicazione on-line fornisce accesso immediato a tutti i possibili utenti. L'Editorial Board multidisciplinare garantisce i requisiti di qualità per la pubblicazione dei contributi.

ISSN 2039-7941

rapporti tecnici INGV

http://istituto.ingv.it/l-ingv/produzione-scientifica/rapporti-tecnici-ingv/

I Rapporti Tecnici INGV pubblicano contributi, sia in italiano che in inglese, di tipo tecnologico e di rilevante interesse tecnico-scientifico per gli ambiti disciplinari propri dell'INGV. La collana Rapporti Tecnici INGV pubblica esclusivamente on-line per garantire agli autori rapidità di diffusione e agli utenti accesso immediato ai dati pubblicati. L'Editorial Board multidisciplinare garantisce i requisiti di qualità per la pubblicazione dei contributi.

ISSN 2039-6651

Miscellanea INGV

http://istituto.ingv.it/l-ingv/produzione-scientifica/miscellanea-ingv/

La collana Miscellanea INGV nasce con l'intento di favorire la pubblicazione di contributi scientifici riguardanti le attività svolte dall'INGV (sismologia, vulcanologia, geologia, geomagnetismo, geochimica, aeronomia e innovazione tecnologica). In particolare, la collana Miscellanea INGV raccoglie reports di progetti scientifici, proceedings di convegni, manuali, monografie di rilevante interesse, raccolte di articoli ecc..

Coordinamento editoriale e impaginazione
Centro Editoriale Nazionale | INGV

Progetto grafico e redazionale
Daniela Riposati | Laboratorio Grafica e Immagini | INGV

Via di Vigna Murata, 605
00143 Roma
Tel. +39 06518601 Fax +39 065041181

http://www.ingv.it

Istituto Nazionale di Geofisica e Vulcanologia