Anno 2011_Numero 195

rapporti tecnici INGV

Nuovo sistema per le localizzazioni automatiche degli eventi sismici basato su tecnologie web: NewWBSM

195

Istituto Nazionale di Geofisica e Vulcanologia

Segreteria di Redazione
Francesca Di Stefano - coordinatore
Tel. +39 06 51860068
Fax +39 06 36915617
Rossella Celi
Tel. +39 06 51860055
Fax +39 06 36915617

redazionecen@ingv.it

ISSN 2039-7941
Anno 2011_Numero 195

rapporti tecnici

INGV

NUOVO SISTEMA PER LE LOCALIZZAZIONI AUTOMATICHE DEGLI EVENTI SISMICI BASATO SU TECNOLOGIE WEB: NEWWBSM

Giovanni Scarpato

INGV (Istituto Nazionale di Geofisica e Vulcanologia, Sezione di Napoli - Osservatorio Vesuviano)

195

Indice

Introduzione

La rapida evoluzione dei sistemi di monitoraggio, legata al generale progresso tecnologico che si è avuto negli ultimi anni, ha consentito lo sviluppo di strumentazione sempre più sofisticata, ha reso disponibili potenti strumenti di calcolo, relativamente a basso costo, e ha aperto nuove prospettive per la trasmissione dei dati, per la loro analisi automatica e per il controllo remoto. I sistemi di nuova generazione sono progettati per ottenere prestazioni sempre più elevate, soprattutto riguardo gli aspetti dell'analisi in tempo reale. Tra questi citiamo *Earthworm* [Johnson et al., 1995], un sistema *open source* sviluppato presso l'*United States Geological Survey* (USGS). I vantaggi offerti da questo tipo di sistema sono molteplici. Esso consente di inviare su rete TCP/IP i segnali provenienti dalle stazioni sismiche verso più centri di raccolta dati, permettendo di eliminare una serie di operazioni manuali che possono comportare ritardi nella comunicazione agli enti competenti dei fenomeni rilevati e favorisce la rapida implementazione di nuovi moduli per l'analisi in tempo reale dei dati. Questo aspetto ha portato come conseguenza un più stretto legame tra l'attività di sorveglianza e l'attività di ricerca in campo sismologico, poiché attualmente la gran parte delle tecniche di analisi sviluppate nell'ambito della ricerca di base, che abbiano una qualche valenza per il monitoraggio, possono essere implementate in maniera automatica ed applicate in tempo reale. Ciò qualifica e rende più efficace il servizio di sorveglianza consentendo di interpretare con maggiore accuratezza i fenomeni che si rilevano. In particolare, con il presente lavoro, si vuole descrivere la funzionalità di *Earthworm* per effettuare localizzazioni degli eventi sismici in maniera automatica. Per questo scopo, la versione di *Earthworm* utilizzata presso la Sezione di Napoli "Osservatorio Vesuviano" dell'INGV si avvale di *Hypoinverse 2000* [Klein, 2002].

Un'ulteriore evoluzione dei moderni sistemi per l'analisi dei dati sismici è rappresentato dall'introduzione di applicazioni web basate su *Java*, il cui pregio principale è quello di poter funzionare su qualunque sistema dotato di *Java Virtual Machine*, indipendentemente dal sistema operativo che si utilizza. Proprio per tale motivo nell'ambito di questo lavoro sono state apportate sostanziali modifiche al sistema *Web Based Seismological Monitoring* (WBSM) [Giudicepietro et al., 2002] implementato presso la Sezione di Napoli "Osservatorio Vesuviano" dell'INGV e finalizzato alla pubblicazione in tempo reale dei risultati del processing dei segnali sismici ed alla visualizzazione, downloading e analisi dei dati via Internet. Questo sistema, oltre a fornire la localizzazione degli eventi sismici, fornisce anche tutti i parametri di qualità calcolati dal programma di localizzazione e consente di visualizzare le forme d'onda, i picking delle fasi ed i relativi residui, offrendo il quadro completo delle informazioni utili alla valutazione della corretta detezione dell'evento ed alla sua classificazione sulla base dell'analisi visiva della forma d'onda. In particolare WBSM sfrutta la tecnologia XML per la rappresentazione dei dati parametrici relativi alle localizzazioni ipocentrali e costituisce un *data base* degli eventi sismici localizzati, comprendente forme d'onda e parametri sintetici, che si aggiorna automaticamente ogni qual volta è localizzato un nuovo evento [Giudicepietro et al., 2002].

Nella prima fase dello sviluppo di WBSM, però, non sono state sfruttate a pieno le potenzialità tipiche delle applicazioni sviluppate in *Java* [Lemay e Cadenhead, 2003], prima tra le quali l'indipendenza dalla piattaforma su cui si esegue l'applicazione, essendo alcuni elementi costitutivi basati su prodotti Microsoft. Infatti, WBSM è stata sviluppata intorno all'utilizzo del *data base* Access, vincolando l'applicazione all'impiego di sistemi Windows.

Con il presente lavoro viene realizzata una nuova versione del sistema, del tutto indipendente dalla piattaforma su cui opera, e parametrizzando la parte di configurazione del sistema stesso, facendo così in modo che ogni modulo sia in grado di leggere, in fase di avvio, una serie di parametri da file di configurazione utili al corretto funzionamento. Ad esempio la directory in cui salvare i file XML, l'IP e la porta su cui rispondere alle richieste *http*, ecc. A tale scopo è stato utilizzato come *data base* MySQL, in grado di girare su una vasta gamma di piattaforme, sono state modificate tutte le procedure di interrogazione al nuovo *data base*, rendendo NewWBSM indipendente dalla piattaforma, e parametrizzate un certo numero di variabili in modo da rendere configurabile il sistema senza dover ricompilare il codice sorgente. NewWBSM è stato realizzato secondo una struttura a livelli logici, come descritto in figura 1. In particolare tre sono i livelli che rappresentano il sistema complessivamente: *dati*, *intermedio* e *presentazione*.

Il primo è il livello che fornisce servizi non direttamente disponibili tramite il Server Web. Questi servizi sono generalmente forniti da applicazioni indipendenti dall'ambiente Web. Tipici esempi di applicazioni presenti a questo livello sono server dati (DBMS), server di mail e delle applicazioni in uso presso l'Osservatorio Vesuviano, quali:

- Sistema Sismometrico Modulare Integrato (SISMI);
- *Earthworm*.

Il secondo è il livello servente, che offre servizi al livello presentazione e dati. In particolare si occupa di ricevere informazioni dal livello dati ed eventualmente inserirle nel *data base*, oppure accogliere le richieste da parte di un operatore per reperire l'informazione dal *data base* e renderla disponibile al livello presentazione. Questo è il livello che si occupa di dare una veste grafica a tutto il contenuto del *data base*.

Figura 1. Schema a blocchi del sistema NewWBSM.

1. Livello dati: *Earthworm*

Earthworm è un sistema modulare per il processamento automatico dei dati sismici che prevede degli strumenti propri per l'acquisizione dati. Tuttavia i dati possono anche essere immessi nel sistema via rete inviandoli con un opportuno protocollo. I moduli di *Earthworm* scambiano dati e messaggi utilizzando un sistema di *broadcasting* basato su *buffer* circolari di memoria condivisa detti *ring* (figura 2).

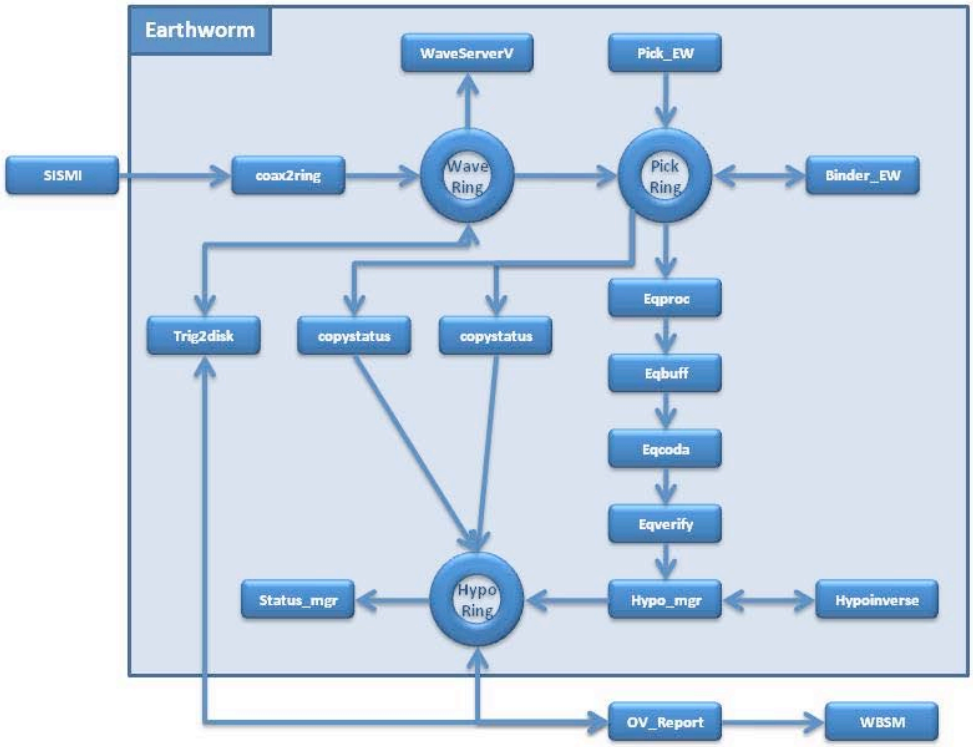


Figura 2. Schema a blocchi di *Earthworm*.

Earthworm è utilizzato presso l'Osservatorio Vesuviano per:

- lo scambio dati tra vari centri di acquisizione intermedi;
- il picking automatico delle fasi;
- la localizzazione automatica degli eventi;
- l'accesso ai dati in tempo reale da remoto;
- l'archiviazione temporanea dei dati su disco;
- l'interfaccia con NewWBSM.

Il diagramma a blocchi di figura 2 mostra uno schema semplificato della architettura di base del sistema.

I cerchi nel diagramma rappresentano i *ring* mentre i rettangoli rappresentano i moduli. *Copystatus* e *Status_mgr* fanno parte del sistema di gestione degli errori di *Earthworm* che permette, una volta rilevato un errore, di mandare mail, visualizzare messaggi o uccidere il modulo sorgente degli errori. Il modulo *Coax2ring* importa i segnali sismici inviatigli via rete da SISMI, sistema distribuito per l'acquisizione e la gestione in tempo reale di dati sismici sviluppato presso il Centro di Monitoraggio dell'Osservatorio Vesuviano [Giudicepietro et al., 2000].

Tre moduli ascoltano il *Wave_Ring*:

- *WaveServerV* che effettua un buffering temporaneo delle forme d'onda acquisite in una coda circolare;
- *Pick_EW* che rileva eventuali fasi sismiche ed invia i picking sul *Pick_Ring*;
- *Copystatus* che semplicemente copia gli eventuali messaggi di errore sull'*Hypo_Ring*.

Tre moduli ascoltano il *Pick_Ring*:

- *Binder_ew* che associa i picking in eventi sismici. Questo modulo ascolta i messaggi prodotti dal *Pick_EW* e produce localizzazioni preliminari.
- *Eqproc* che è il primo di una serie di sottomoduli che effettuano la localizzazione dell'evento sismico. I sottomoduli sono collegati fra loro tramite delle *pipe* e sono visti da *Earthworm* come un unico modulo. *Eqproc* assembla la localizzazione preliminare del *Binder_ew* con le relative fasi prodotte da *Pick_EW* e manda l'evento così assemblato a *eqbuf*. *Eqbuf* gestisce una coda di eventi verso il successivo processo di elaborazione che è formato da *eqcoda* che calcola le durate degli eventi, *eqverify* che effettua una serie di controlli per eliminare eventi spuri, *Hypo_mgr* che effettua la localizzazione finale dell'evento e la invia sull'*Hypo_Ring*.

Tre moduli ascoltano l'*Hypo_Ring*:

- *Status_mgr* che è il modulo che si occupa di gestire i messaggi di errore del sistema;
- *Trig2disk* che ha il compito di prelevare attraverso il *WaveServerV* le forme d'onda dell'evento sismico;
- *OV_Repor*t che è un modulo sviluppato nell'ambito del presente lavoro per interfacciare NewWBSM con *Earthworm*. Tale modulo è utilizzato per inviare l'evento sismico rilevato, completo delle informazioni di localizzazione, picking e forme d'onda, al sistema remoto di archiviazione/elaborazione di NewWBSM.

1.1 Piattaforma

Java, diversamente da altri linguaggi di programmazione, compila i sorgenti dei suoi programmi in un codice detto *Bytecode*, diverso dal linguaggio della macchina su cui è compilato, mentre linguaggi come il *C++* compilano i sorgenti dei programmi in un codice che è il codice della macchina (per macchina si intende computer + sistema operativo) su cui è eseguito. Quindi, per eseguire un programma *Java*, occorre avere uno strumento chiamato *Java Virtual Machine* (JVM), il quale interpreta il bytecode generato dal compilatore *Java* e lo esegue sulla macchina su cui è installato. Grazie alla *Java Virtual Machine*, *Java* è indipendente dalla piattaforma. Infatti il programma compilato *Java* è legato alla JVM e non al sistema

operativo, per cui sarà possibile eseguire lo stesso programma *Java* su una piattaforma Windows e su una piattaforma Linux, purché su detti sistemi sia installata una *Java Virtual Machine*.

La JVM è implementata generalmente nei vari Browser (come Mozilla Firefox, Google Chrome, Internet Explorer, ecc.) per poter eseguire i programmi *Java* presenti nella rete: i cosiddetti *Java Applet*.

1.2 *Data base*

MySQL, scelto per realizzare NewWBSM, è un *Database Management System* (DBMS) relazionale, compatibile con la maggior parte dei sistemi operativi, tra cui sistemi Unix, Linux o Windows, anche se prevale un suo utilizzo in ambito Unix o Linux [Harms e Petreley, 2001].

La struttura fondamentale del modello relazionale e' appunto la "relazione", cioè una tabella bidimensionale costituita da righe e colonne. Le relazioni rappresentano le entità che si ritiene essere interessanti nel database. Ogni istanza dell'entità troverà posto in una tupla della relazione, mentre gli attributi della relazione rappresenteranno le proprietà dell'entità.

Insieme al modello relazionale è stato introdotto il linguaggio SQL (*Structured Query Language*), che consente di operare sui dati tramite frasi che contengono parole chiave prese dal linguaggio corrente (ovviamente della lingua inglese).

Naturalmente, visto l'ampio successo dei database relazionali, sono molti gli RDBMS presenti sul mercato: IBM DB2, Oracle, Microsoft SQL Server, Sybase, Filemaker Pro, Microsoft Access, Informix, PostgreSQL, SQLite, oltre naturalmente a MySql, sviluppato dalla compagnia svedese MySQL AB. Alcuni di questi sono software proprietari, mentre altri fanno parte della categoria open source: questi ultimi, fra quelli citati, sono MySql, PostgreSQL e SQLite.

MySQL permette di utilizzare diversi "*storage engine*" (motori di archiviazione) per la memorizzazione dei dati, che si distinguono principalmente in *transazionali* e *non transazionali*.
I motori transazionali offrono alcuni importanti vantaggi: sono più sicuri (permettono di recuperare i dati anche in caso di crash di *MySQL* o di problemi hardware) e consentono di effettuare più modifiche e convalidarle tutte insieme o, al contrario, ripristinare la situazione preesistente se qualcosa va male.

Dal canto loro, i motori non transazionali hanno il vantaggio di una maggior velocità, minore utilizzo di spazio su disco e minor richiesta di memoria per gli update. È anche possibile combinare tabelle transazionali e non nelle stesse istruzioni, anche se, in questo caso, le modifiche fatte sulle tabelle non transazionali divengono comunque effettive nel momento in cui sono eseguite.

Quando si crea una tabella si specifica a MySQL di che tipo si tratta attraverso l'opzione ENGINE:

```
CREATE TABLE tabella (a INT) ENGINE = INNODB;
```

Nel caso in cui la dichiarazione venga omessa, *MySQL* utilizzerà il tipo di default, che normalmente è *MyISAM*.

Nel sistema che si sta descrivendo, è stato utilizzato *MySQL* nella forma *InnoDB*. Questo è uno *storage engine transazionale* dotato di capacità di *commit*, *rollback* e *crash recovery*. È ottimizzato per l'uso concorrente dei dati fra molti utenti e per essere molto performante anche su grandi quantità di dati e consente l'uso delle transazioni.

In figura 3 è descritto lo schema logico del database creato per il sistema in oggetto. La relazione tra le tabelle "evento" e "stazioni" viene realizzata nel livello "Presentazione". Nel caso di una eventuale localizzazione, esiste una associazione tra *singolo evento sismico* ed *N stazioni sismiche*. Tramite la servlet *eqview.class*, del livello "Presentazione", viene effettuata al sistema la richiesta di visualizzare le informazioni associate all'evento sismico e ad esse messe in relazione tutte le informazioni associate alle stazioni sismiche cha hanno contribuito alla detezione dell'evento stesso.

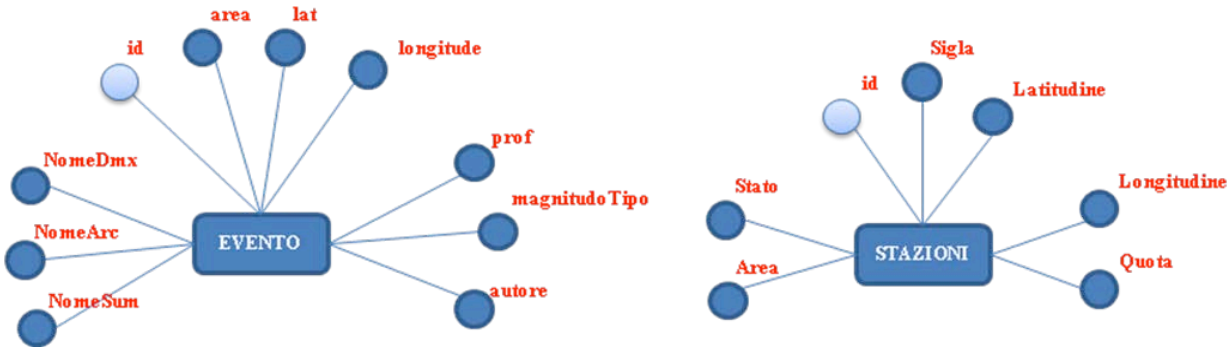


Figura 3. Diagramma ER del database "*real_time_location*", con i principali attributi.

In figura 4 si può notare la struttura del *data base* di NewWBSM. Il *data base* è *real_time_location* ed è costituito da due tabelle. La prima contiene tutte le informazioni utili riguardo la rete sismica, come ad esempio la sigla delle stazioni, la loro posizione GPS (Global Position System), la quota dal livello del mare, lo stato di funzionamento, ecc.

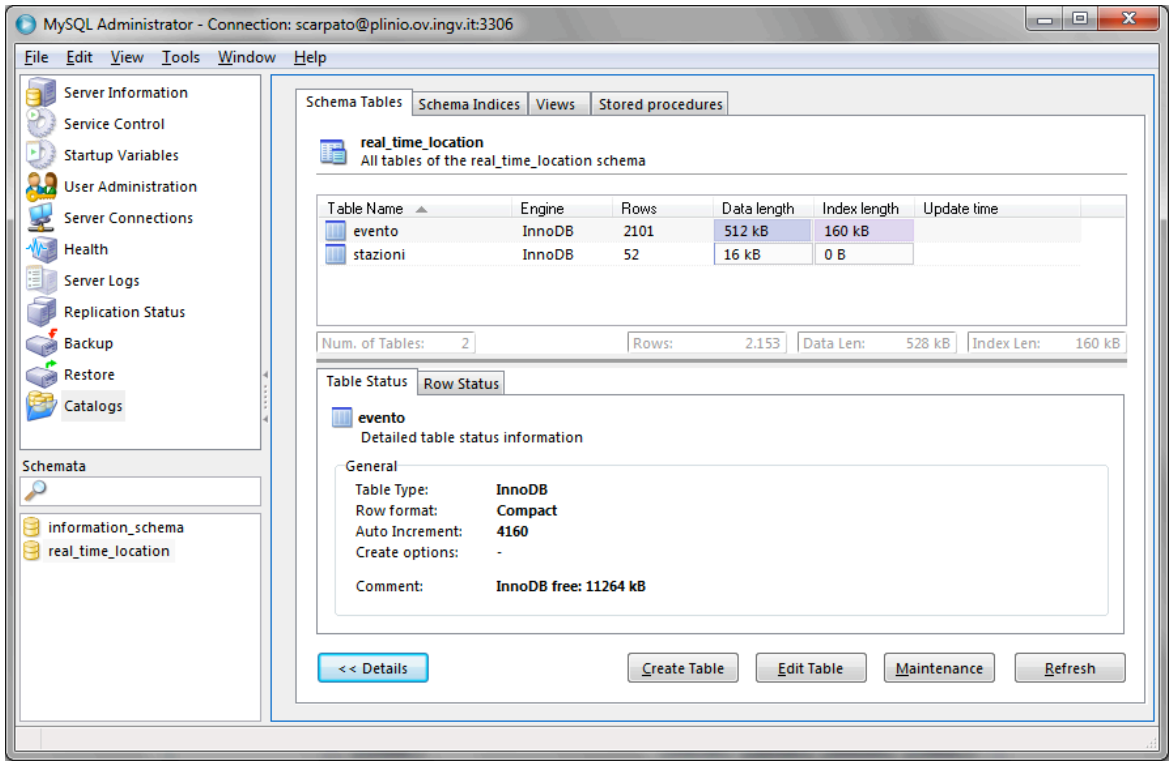


Figura 4. Tipo di motore usato nella costruzione del Data Base di NewWBSM.

In figura 5 è illustrata la tabella "*stazioni*" contenente le informazioni dettagliate riguardanti le singole stazioni sismiche che compongono la rete sismica di monitoraggio, quali ad esempio il nome o sigla della stazione sismica, la posizione GPS, composta da latitudine e longitudine, lo stato della stazione, in modo da evidenziare se questa è attiva o meno, ecc.

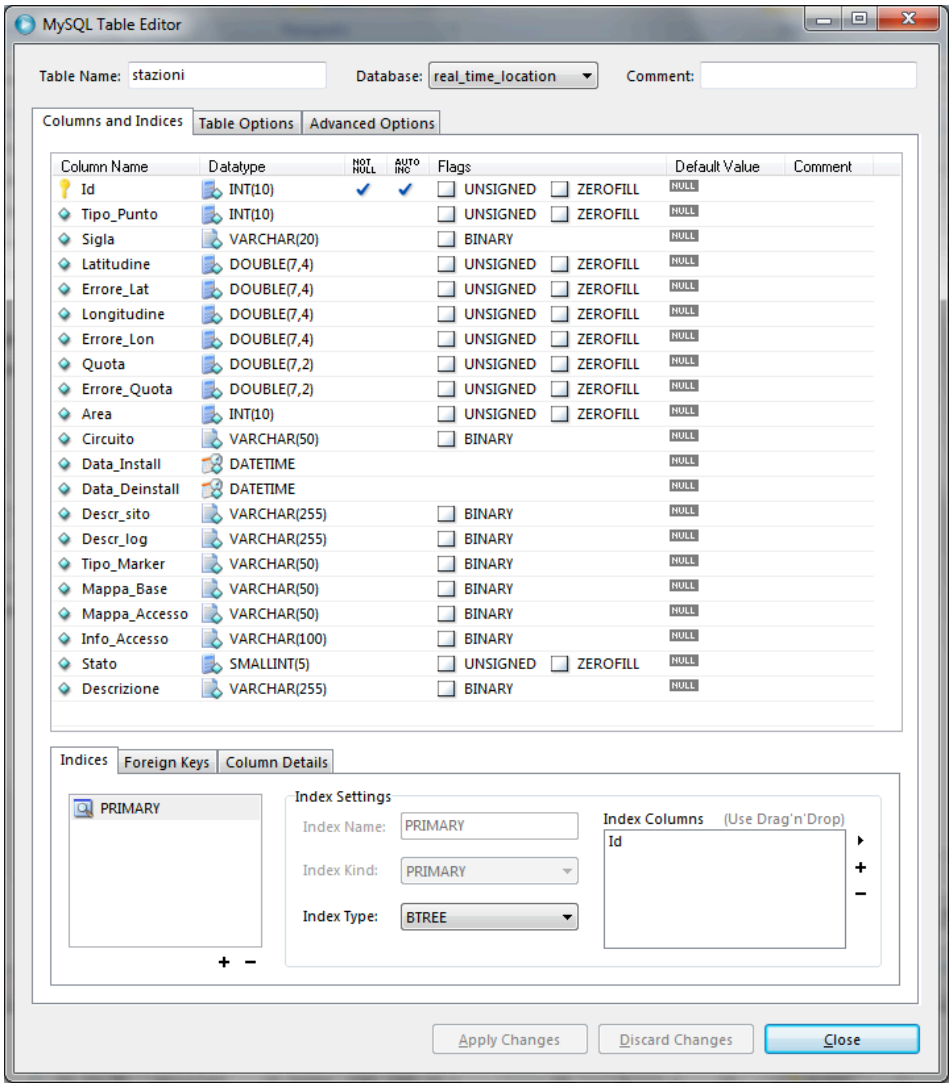


Figura 5. Struttura della tabella delle stazioni.

In figura 6 è illustrata la tabella "*evento*" contenente le informazioni riguardanti gli eventi sismici localizzati automaticamente dal sistema NewWBSM, quali ad esempio il tempo di arrivo dell'evento sismico, l'area vulcanica interessata dall'evento stesso, la magnitudo, ecc.

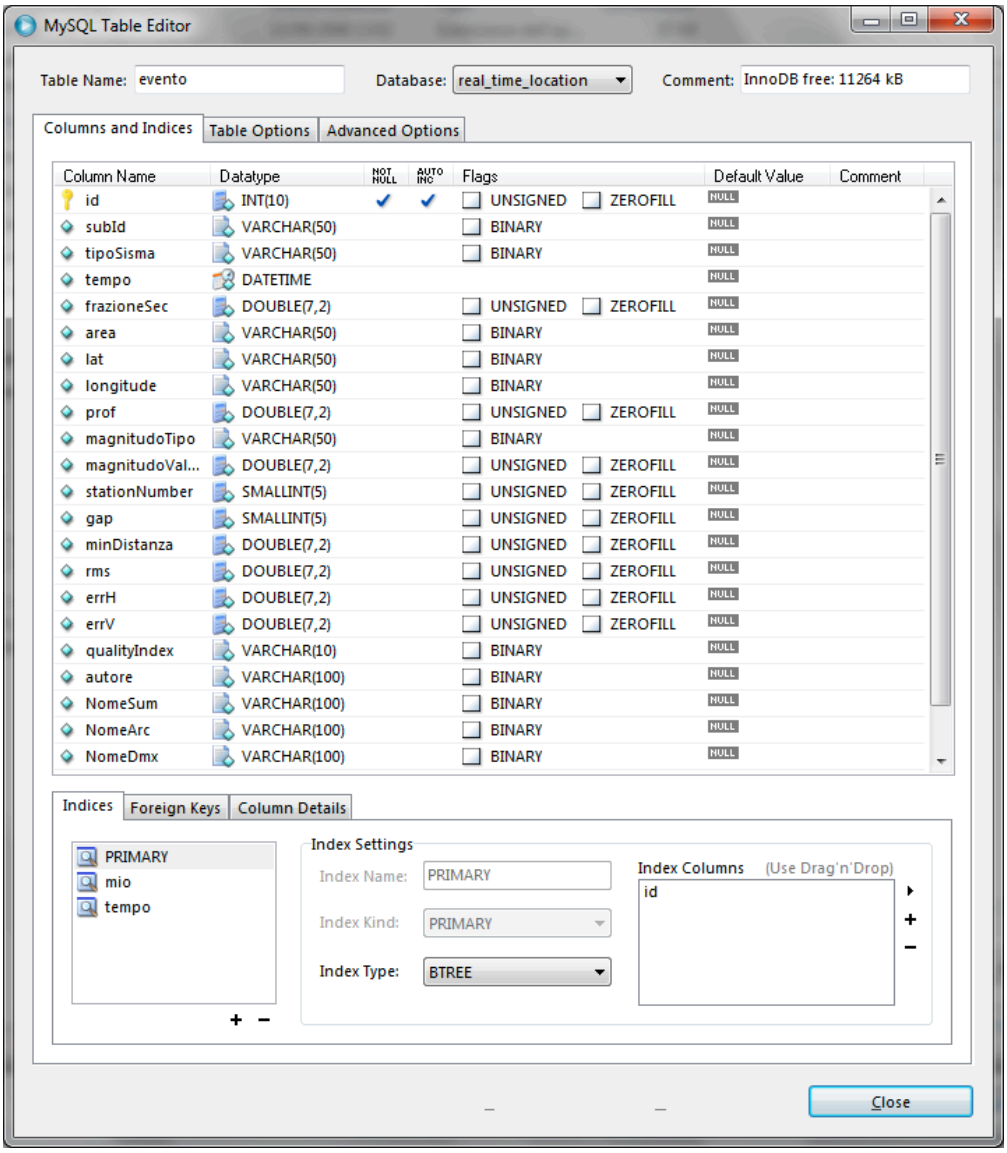


Figura 6. Struttura della tabella degli eventi sismici.

In figura 7 è possibile consultare la lista di un certo numero di eventi sismici localizzati dal sistema, da cui si possono ricavare tutte le informazioni sulla localizzazione, quali tempo origine, area epicentrale, latitudine e longitudine dell'epicentro, profondità, ecc.

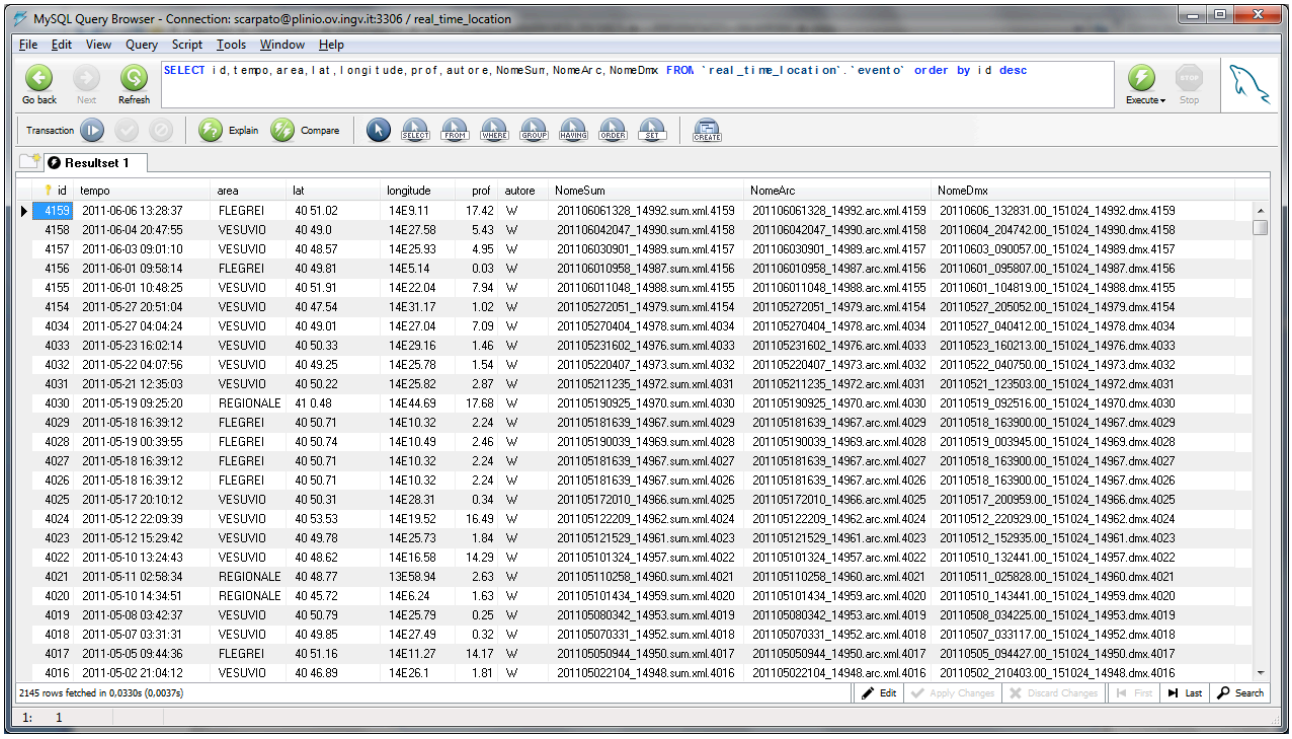


Figura 7. Lista ultimi eventi sismici localizzati dal sistema e inseriti nella tabella "evento".

2. Livello intermedio

Il livello intermedio è il livello che contiene la logica servente del sistema. Esso è in grado di soddisfare le richieste di dati e di elaborazione del *client*. Le modalità di realizzazione del livello intermedio dipendono spesso dalle caratteristiche e dalle tecnologie supportate dal server Web e/o da componenti installati sul server applicativo. In ogni caso la funzionalità fondamentale del server Web su cui si basa l'intera applicazione è il supporto alle elaborazioni. Per la realizzazione del sistema in oggetto si è scelto di utilizzare *Tomcat*, un *servlet container* usato per lo sviluppo di tecnologie, quali *Java Servlet* e *Java Server Page*. Esso è realizzato sotto la *Apache Software License* ed è sviluppato in maniera aperta dalla collaborazione di sviluppatori di software di tutto il mondo. Il livello intermedio, quindi, è costituito da un insieme di script, componenti e programmi interagenti tra di loro e con il server Web, tra cui *Java Servlet*, che consentono di eseguire classi *Java* su richiesta del *client* (portabile su qualsiasi piattaforma).
Una *Servlet*, nella sua forma più generale, è un'istanza della classe che implementa l'interfaccia *Javax.servlet.Servlet* , e cioè *GenericServlet* del package *Javax.servlet*, con un proprio ciclo di vita; ma come è stato già detto quella più utilizzata è quella che usa il protocollo HTTP e che si costruisce estendendo la classe *Javax.servlet.http.HttpServlet*.

Di seguito viene analizzato il tipico ciclo di vita di una *Servlet*. All'avvio, il server carica in memoria la classe *Servlet* ed eventualmente le classi utilizzate da questa, e crea un'istanza chiamando il costruttore senza argomenti. Subito dopo viene chiamato il metodo *init*(), tramite il quale vengono allocate tutte le risorse che occorrono per il corretto funzionamento della *servlet*. Infatti è con tale metodo che avvengono le inizializzazioni delle variabili globali, procedura che viene fatta una sola volta durante l'intero ciclo di vita della *Servlet*. Infine viene caricato l'oggetto *ServletConfig* che potrà poi essere recuperato più tardi mediante il metodo *getServletConfig()*. L'oggetto *ServletConfig* contiene i parametri della *Servlet*, nonché il riferimento a *ServletContext*, che rappresenta il contesto in cui gira la *Servlet*.

Una volta inizializzata la *Servlet*, viene chiamato il metodo *service (ServletRequest req, ServletResponse res)* per ogni richiesta che arriva dai *client*; questo metodo viene chiamato in modo concorrente, cioè più *Thread* possono invocarlo nello stesso momento. In casi particolari e cioè quando si lavora con risorse non condivisibili, è possibile implementare *Servlet* non concorrenti. Quando una *Servlet* deve essere arrestata, ad esempio se deve essere aggiornata o bisogna riavviare il server, viene chiamato il

metodo *destroy()*, nel quale vengono rilasciate le risorse allocate nel metodo *init*. Anche questo metodo viene chiamato una sola volta durante il ciclo di vita della *Servlet*.

2.1 Modulo invio dati: OV_Report

Il modulo di interazione con *Earthworm*, è integrato in *Earthworm* stesso, in modo che può essere gestito e controllato dai processi preposti al management generale del sistema.
Questo modulo, denominato *OV_report*, svolge alcune operazioni che possono essere suddivise in principali e secondarie. Le funzioni principali sono:

- sniffare l'*Hypo_ring* per rilevare la presenza di eventuali eventi sismici localizzati dal sistema automatica di *Earthworm*;
- creare le strutture dati rappresentative dell'evento sismico ed inviarle al processo di traduzione-spedizione;
- seguire l'andamento del processo di spedizione-traduzione.

Le funzioni definite secondarie sono:

- gestire l'eventuale caduta, anche per diversi giorni, del servizio di traduzione-spedizione senza alcuna perdita di informazioni;
- registrare in un file *log* le attività legate alle operazioni del modulo durante il suo funzionamento;
- visualizzare messaggi di *debug* utili ad individuare eventuali anomalie nell'esecuzione del modulo.

Nella realizzazione delle funzioni citate è necessario che i compiti secondari siano svolti in parallelo alla funzione di captazione degli eventi sismici, la quale deve essere sempre attiva, altrimenti il modulo di interazione rischia di perdere l'acquisizione di un evento sismico localizzato da *Earthworm*. Inoltre è necessario limitare il tempo di utilizzo della CPU per evitare un rallentamento o addirittura un blocco nelle esecuzioni dei moduli di *Earthworm* preposti alla localizzazione automatica degli eventi sismici con conseguente perdita di dati. In base a queste considerazioni *OV_report* è stato strutturato come mostrato in figura 8.

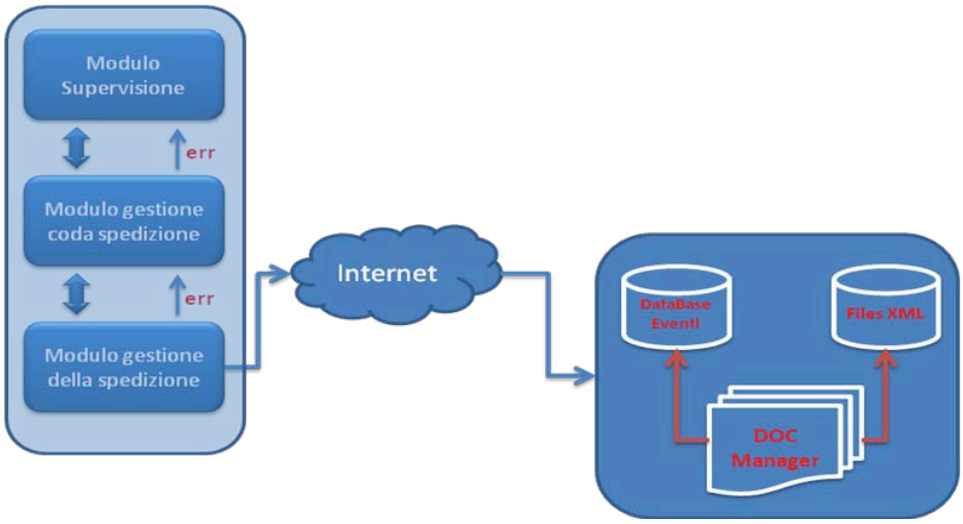


Figura 8. Architettura del modulo di interazione con *Earthworm* (*OV_report*).

La principale attività del modulo di supervisione è quella di controllare continuamente l'arrivo di un nuovo evento sismico generato dal sistema di localizzazione automatica. Quest'attività è realizzata interrogando il *ring* dedicato agli eventi. Nel caso in cui l'interrogazione ha esito positivo le strutture dati

lette sono aggiunte alla coda di spedizione. In questa fase viene effettuata una chiamata al modulo *sendhttp.class*. In figura 9 è descritta la sequenza temporale delle funzioni svolte dal modulo. Dopo la fase di inizializzazione, in cui viene letto un file di configurazione con una serie di parametri quali directory dei file .arc, .sum, .dmx e degli equivalenti in formato .xml, vengono tradotte le informazioni generate dai moduli di *Earthworm* in formato *xml* ed inviate via *http* al server web *Tomcat*.

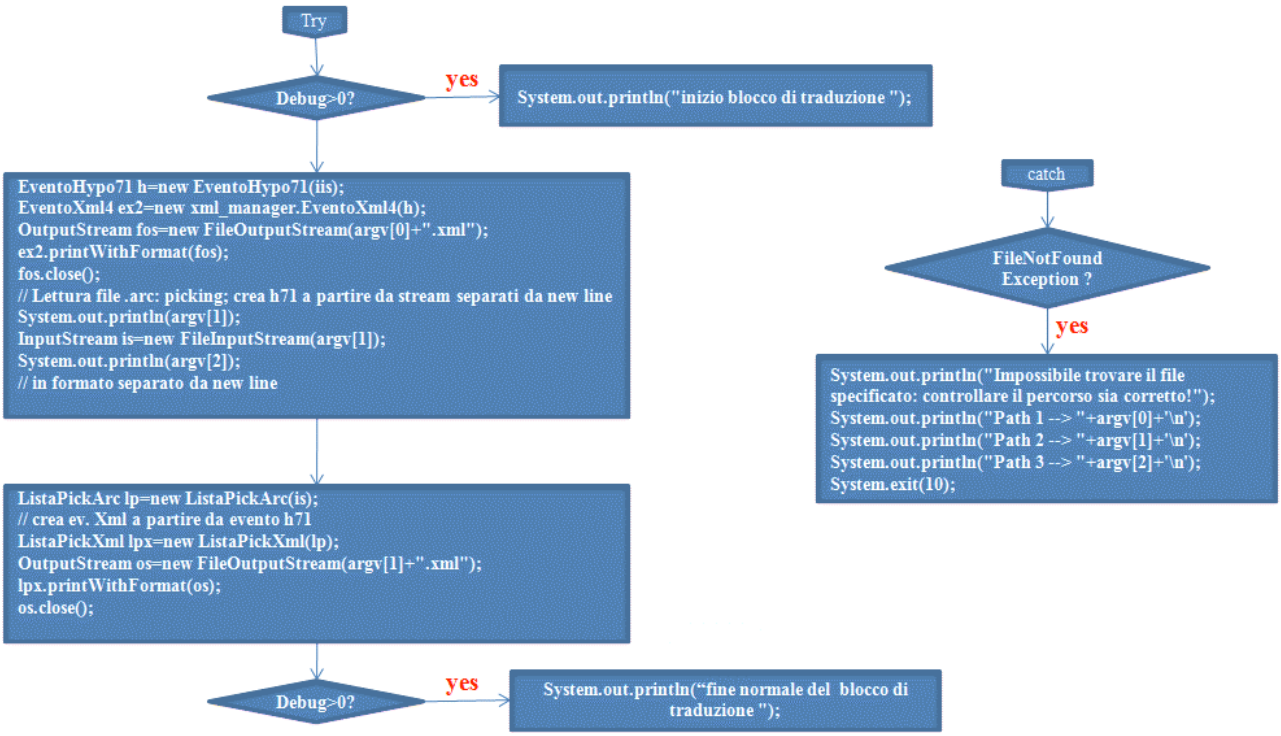


Figura 9. Diagramma in UML del modulo *sendhttp.class*.

2.2 Modulo ricezione ed inserimento dati

Il modulo di ricezione ed inserimento è realizzato tramite la *servlet ParserServ.Java.* Affinché il livello applicazione possa interagire con il livello *data base*, è stata creata una libreria che include un certo numero di funzioni. In questo modo è possibile inserire, aggiornare e reperire informazioni dal *data base*. Il codice per realizzare tale libreria è scritto in *Java* e modificato, rispetto alla versione precedente, secondo la sintassi *SQL*.

ParserServ ha il compito di effettuare un *parsing* sulla struttura dati ricevuta via *http* da *OV_Report* ed eventualmente inserire le informazioni in essa contenute nelle apposite tabelle del *data base*.
L'operazione di inserimento consiste nei seguenti passi:

- apertura della sessione DB di inserimento con il *data base*;
- formattazione della query, del tipo *INSERT INTO tbl_name (col1,...,colN) VALUES(val1,...,valN);*
- *flushing* della query su socket TCP;
- invio delle strutture XML di *summary*, *archive* e *dmx*, prodotte da Hypo2000, affinchè vengano salvate su hard disk;
- chiusura della sessione DB.

In figura 10 è descritta la fase di inizializzazione, in cui vengono stabilite le directory in cui salvare le strutture dati xml.

```
public void init(ServletConfig config) throws ServletException{
        super.init(config);
        ResourceBundle param = ResourceBundle.getBundle("configurazione");
        UrlPort = param.getString("uploadDir");
        String dirName=UrlPort;
        this.log("Directory di Destinazione: "+dirName);
        if (dirName == null){
                throw new ServletException("Fornire correttamente il parametro "+ UrlPort);
        }
        dir = new File(dirName);
        if (! dir.isDirectory()){
                throw new ServletException("La directory "+dirName+" non e' valida!");
        }
        arcDir=new File(dir,"arc");
        if (!arcDir.exists())
                arcDir.mkdir();
        dmxDir=new File(dir,"dmx");
        if (!dmxDir.exists())
                dmxDir.mkdir();
        sumDir=new File(dir,"sum");
        if (!sumDir.exists())
                sumDir.mkdir();
        counter = 0;
        v= new Vector();
}
```

Figura 10. Fase di inizializzazione modulo.

In figura 11 è descritta una parte del codice in cui viene effettuato il *parsing* delle strutture dati xml, per l'eventuale validazione, ed effettuato l'inserimento in *data base* delle informazioni contenute nelle strutture. Tale inserimento avviene avvalendosi di librerie scritte ad - hoc per tutte le operazioni di *insert*, *update*, *delete*, ecc. necessarie all'interazione con il *data base*.

```
        ListaPickXml lpx=new  ListaPickXml(stream ,"arc Xml");
          if (lpx==null){
                System.out.println("ERRORE\n c'è un problema nella creazione della DOM di
 :"+fileNameLpx);
                out.println("ERRORE\n c'è un problema nella creazione della DOM di
 :"+fileNameLpx);
                return;
          }
                ……
          if (!(((part = mp.readNextPart())!=null) &&(part.isFile()))){
                out.println("ERRORE\n mi aspetto un atro parametro file");
                ...
                return;
          }
          name = part.getName();
          if (!name.equals("dmx")){
                out.println("ERRORE\n il nome del file non è dmx");
                System.out.println("ERRORE\n il nome del file non è dmx");
                return;
          }
          filePart = (FilePart) part;
          String fileNameDmx = filePart.getFileName();
          if (fileNameDmx==null){
                out.println("ERRORE"+'\n'+" il nome del file nel file system è vuoto");
                System.out.println("ERRORE"+'\n'+" il nome del file nel file system è vuoto");
                return;
          }
          // part attualmente contiene le forme d'onde in dmx
                ……..
          try{
                db=new Db(dir,arcDir,dmxDir,sumDir);
          } catch (Exception e){
                ...
          }
          //inserimento dati nel DB e contestuale scrittura dei 3 file dell'evento
          long count=db.insert(ex,fileNameEx,lpx,fileNameLpx,filePart,fileNameDmx);
          if (count>0){
                out.println("-COM- \n");
                if (debug>0)
                out.println(" 3 file ricevuti e processati e codiceDB="+count);
          }
          while(mp.readNextPart()!=null) ;  //consumo i restanti caratteri
          out.flush();
          out.close();
```

Figura 11. Frammento di codice per il parsing e l'inserimento in DB.

In figura 12 è illustrato il diagramma temporale della funzione insert, in cui si notano le seguenti fasi:

- Preparazione della connessione socket al *data base*, in cui *url*, *utenteDb* e *pwdDb* sono parametri il cui valore viene letto dal file di configurazione del sistema e rappresentano rispettivamente;
- Preparazione della query da effettuare al *data base*, del tipo:
- *insert into " + dbTabellaEventi + " (subId,tempo , area, lat, longitude, prof, magnitudoTipo, magnitudoValore, qualityIndex, autore, nomeSum, nomeArc, nomeDmx) " values (" valore1", … , "valoreN");*
- Invio query al *data base* con l'istruzione *int i=stmt.executeUpdate(SQLquery)*.

Figura 12. Funzione "*insert*".

3. Livello presentazione: Interfaccia Utente

In figura 13 è illustrata la pagina HTML che rappresenta l'interfaccia grafica al sistema, ovvero al *data base* in cui sono salvate le informazioni delle localizzazioni automatiche.

Tramite un qualunque browser web, accedendo al link *http://plinio.ov.ingv.it:8080/mywbsm/Eqvedi?id=-1* viene attivata la *servlet EqView.Java*, il cui alias per Tomcat è *Eqvedi*. Il parametro id=-1 indica a tale *servlet* di visualizzare l'ultimo evento disponibile nel *data base*.

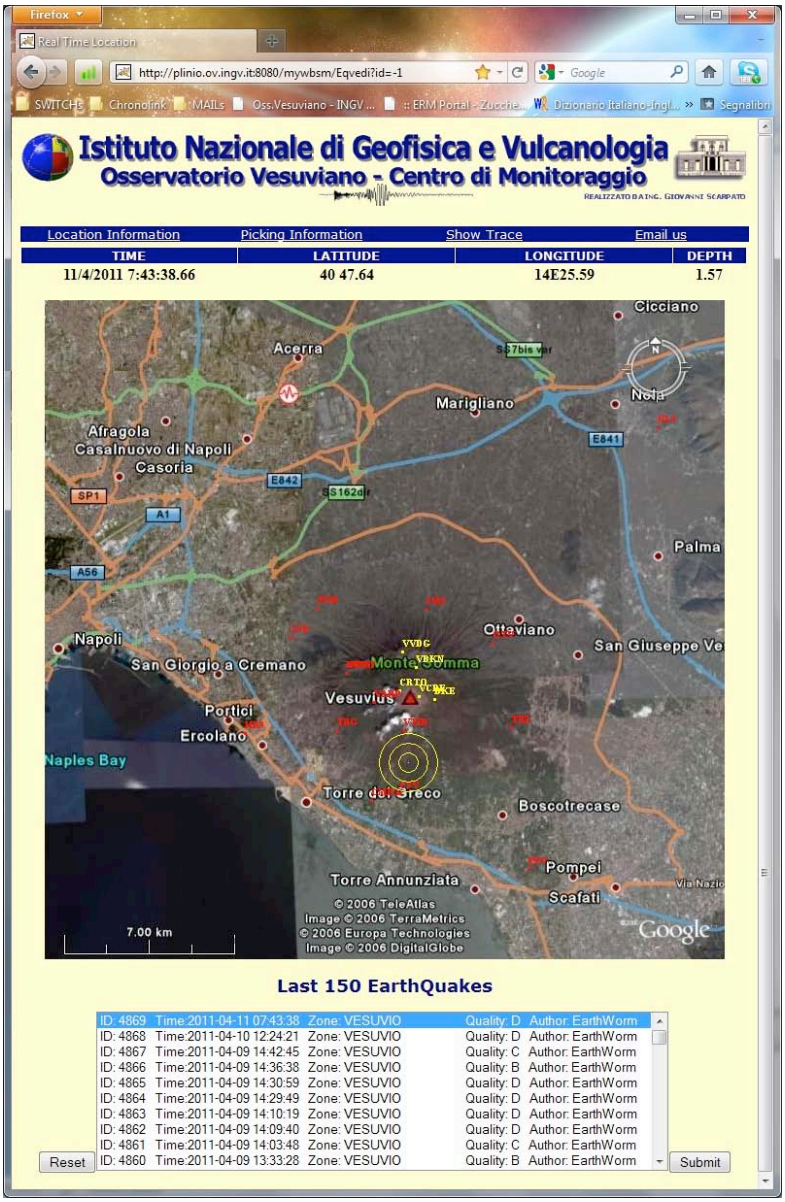


Figura 13. Interfaccia web relativa ad un esempio di localizzazione.

Nelle figure 14, 15 e 16 sono descritte le operazioni compiute affinchè sia possibile visualizzare tutte le informazioni relative ad una certa localizzazione automatica, quali la lista delle stazioni che compongono la rete sismica, evidenziando quelle che hanno contribuito alla localizzazione, l'epicentro dell'evento, sia su mappa con un'animazione che testuale, il tempo origine ed il link ad altre *servlet* che forniscono informazioni più estese. In particolare, dopo la consueta lettura di un file di configurazione da cui estrarre parametri modificabili dall'amministratore del sistema, viene effettuata una query al *data base* per estrarre l'ultimo evento sismico. Successivamente con una seconda query vengono estratte dalla tabella stazioni tutte le stazioni attive della rete sismica.

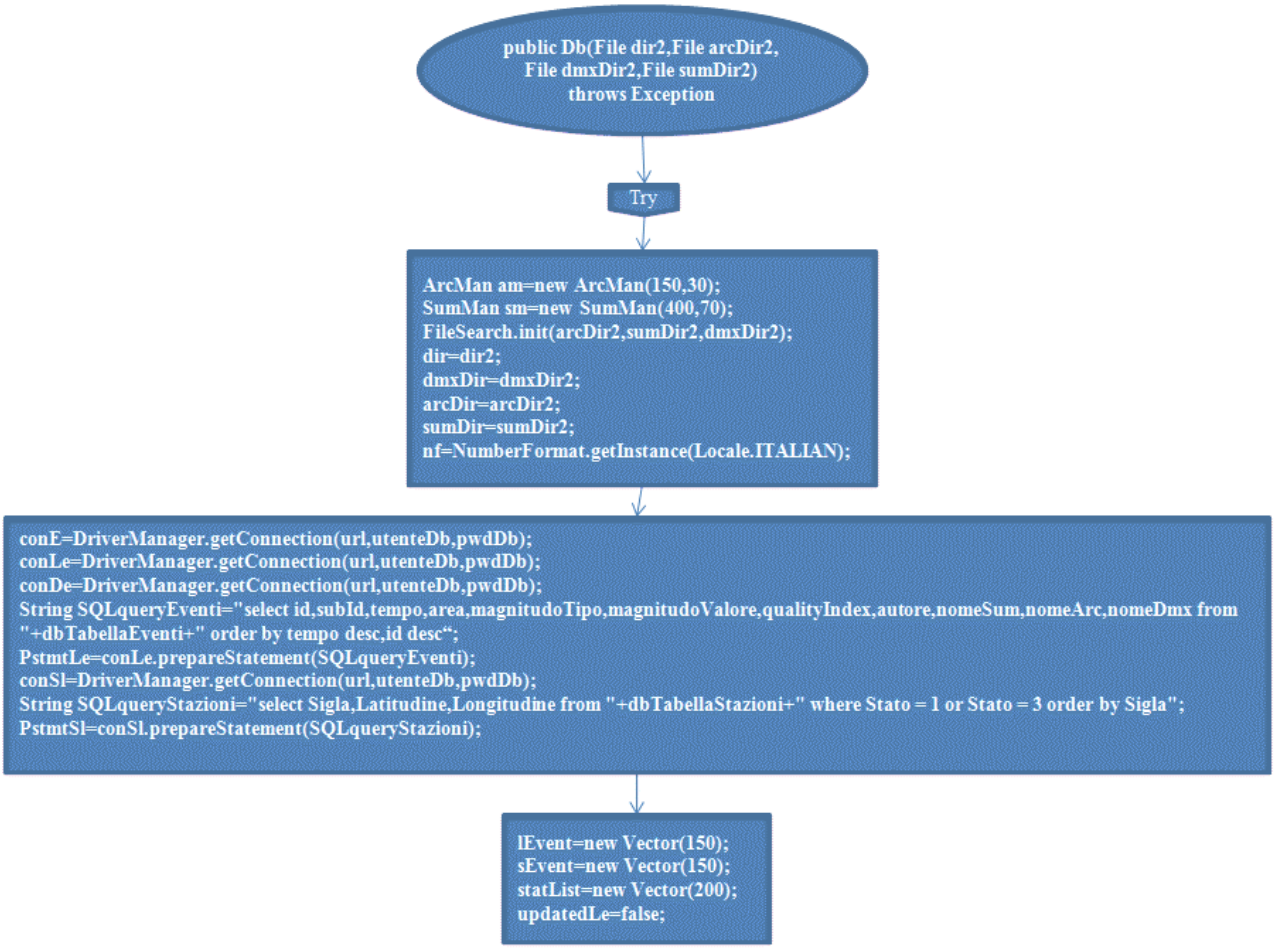


Figura 14. Fase di preparazione per l'accesso al *data base*.

Infine viene effettuata l'ultima query con la quale sono estratte tutte le informazioni di *summary* relative all'evento sismico selezionato, quali tempo origine, latitudine e longitudine dell'evento, ecc.

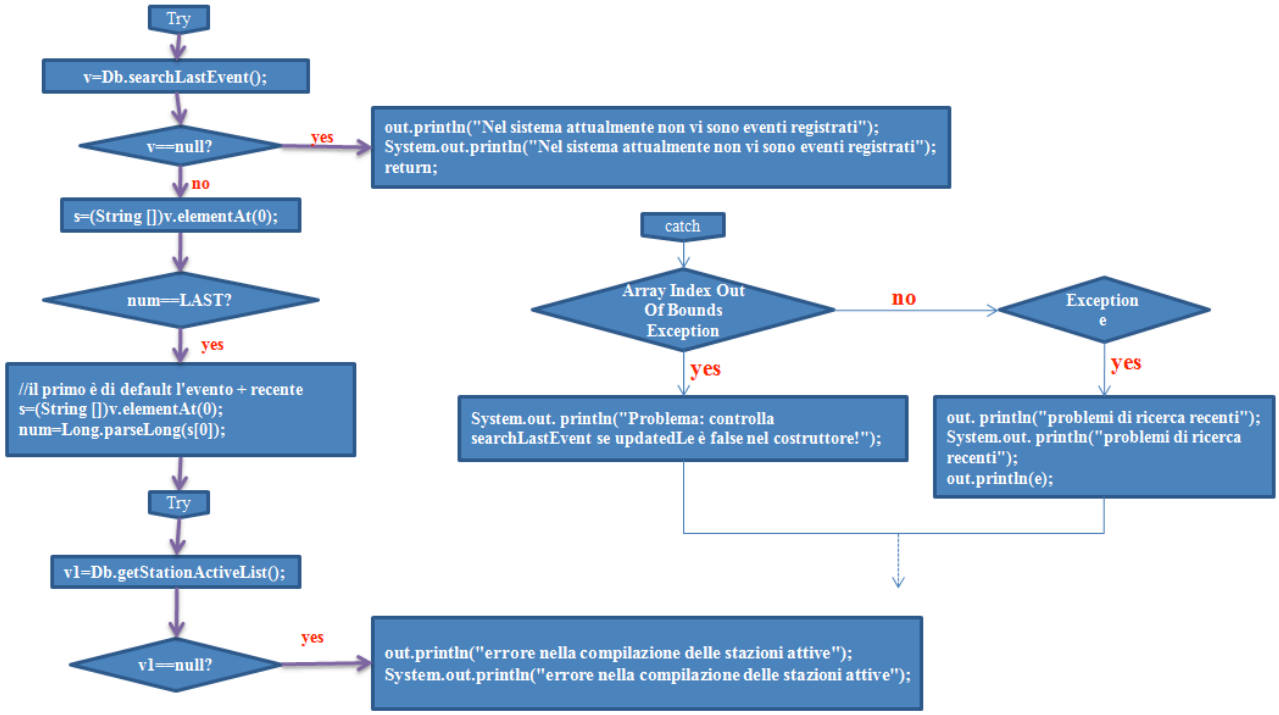


Figura 15. Funzioni *searchLastEvent* e *getStationName*.

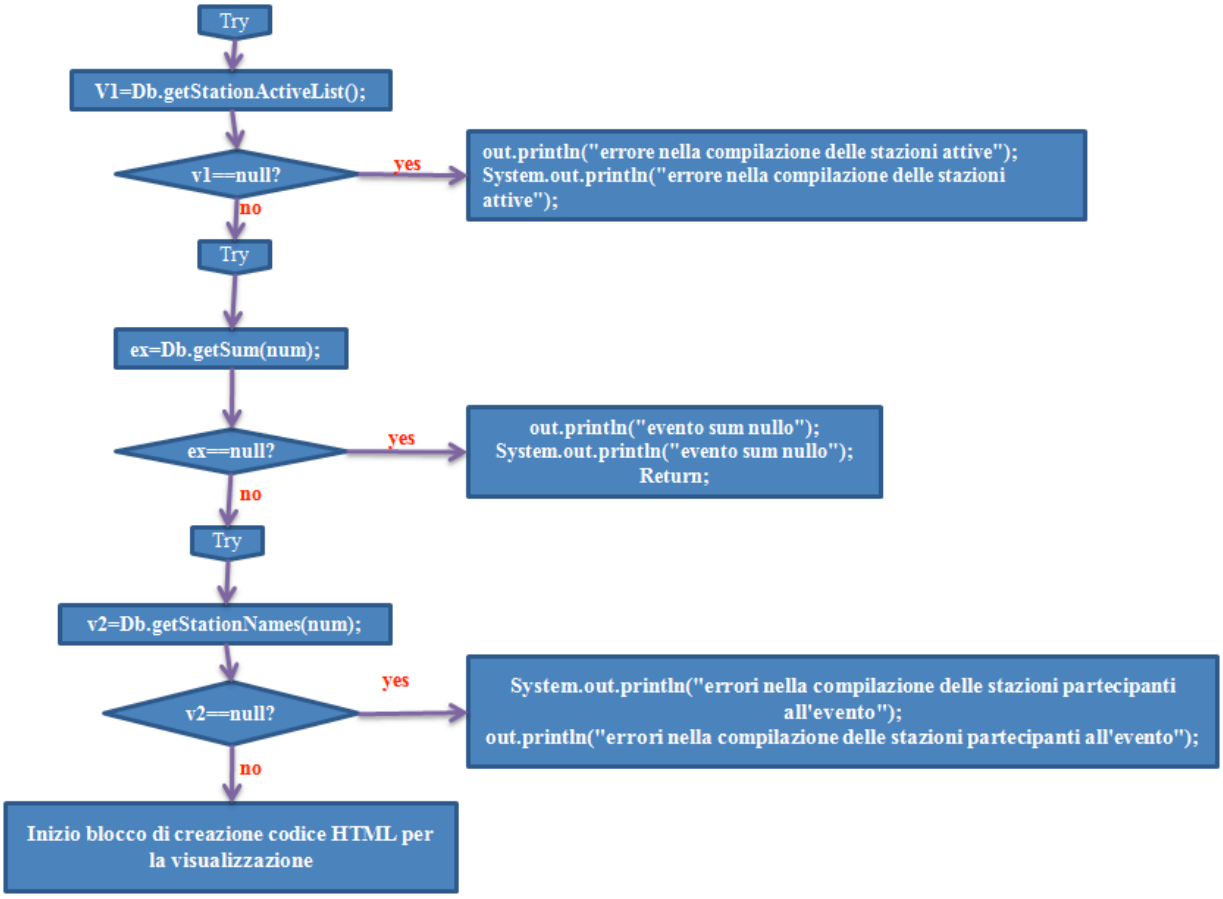
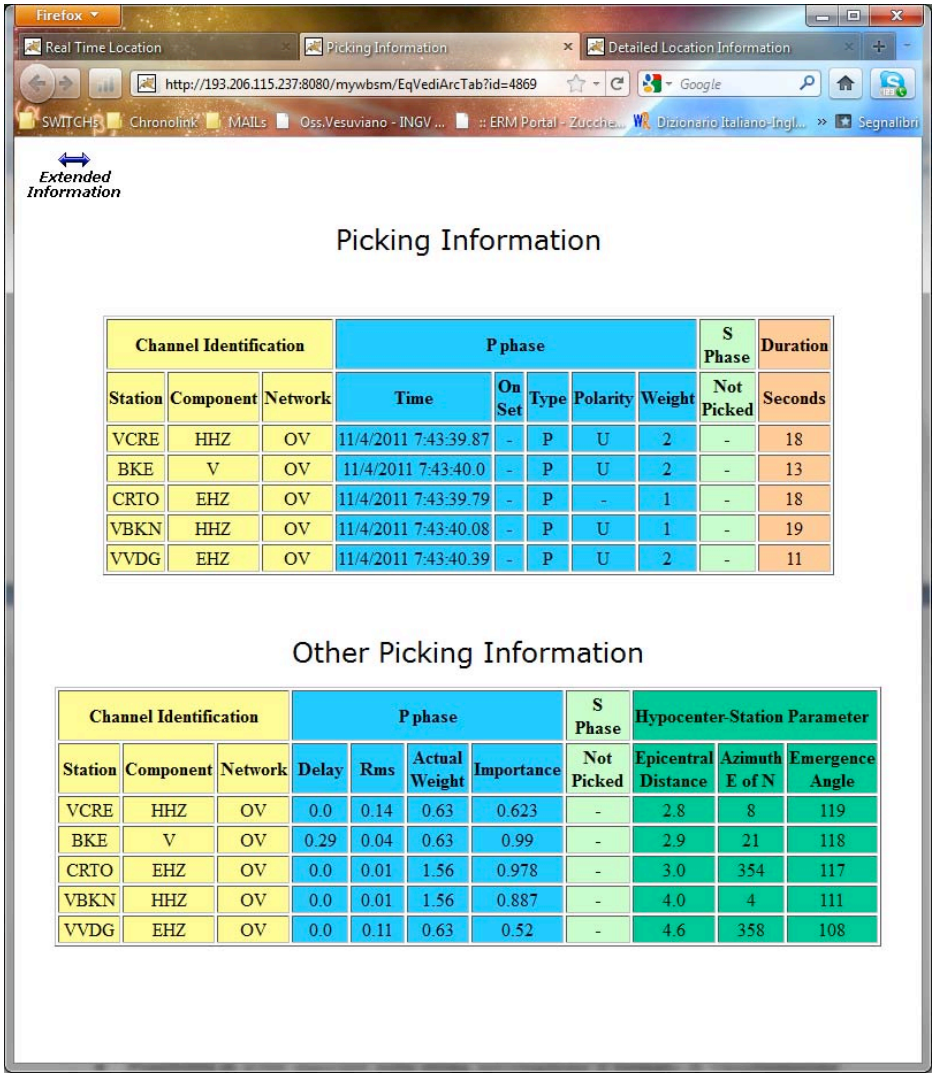


Figura 16. Funzione di ricerca delle stazioni attive.

In figura 17 è illustrata l'interfaccia HTML, generata da un'apposita *servlet*. Tali informazioni sono estratte dal file XML che traduce il contenuto "*archive*" della localizzazione effettuata da Earthworm. Da questa pagina si possono leggere tutte le informazioni legate al *picking* dell'evento sismico, quali le terne SCN (*Station-Component-Network*), i tempi di arrivo alle stazioni, la polarità, l'RMS, ecc.



Channel Identification			P phase					S Phase	Duration
Station	Component	Network	Time	On Set	Type	Polarity	Weight	Not Picked	Seconds
VCRE	HHZ	OV	11/4/2011 7:43:39.87	-	P	U	2	-	18
BKE	V	OV	11/4/2011 7:43:40.0	-	P	U	2	-	13
CRTO	EHZ	OV	11/4/2011 7:43:39.79	-	P	-	1	-	18
VBKN	HHZ	OV	11/4/2011 7:43:40.08	-	P	U	1	-	19
VVDG	EHZ	OV	11/4/2011 7:43:40.39	-	P	U	2	-	11

Channel Identification			P phase				S Phase	Hypocenter-Station Parameter		
Station	Component	Network	Delay	Rms	Actual Weight	Importance	Not Picked	Epicentral Distance	Azimuth E of N	Emergence Angle
VCRE	HHZ	OV	0.0	0.14	0.63	0.623	-	2.8	8	119
BKE	V	OV	0.29	0.04	0.63	0.99	-	2.9	21	118
CRTO	EHZ	OV	0.0	0.01	1.56	0.978	-	3.0	354	117
VBKN	HHZ	OV	0.0	0.01	1.56	0.887	-	4.0	4	111
VVDG	EHZ	OV	0.0	0.11	0.63	0.52	-	4.6	358	108

Figura 17. Informazioni sui *Picking*.

In figura 18, un'ulteriore *servlet* traduce tutte le informazioni contenute nel file XML, traduzione del *summary* di *Earthworm*. Tale *servlet* genera un codice HTML con le informazioni legate alla localizzazione, come latitudine e longitudine dell'epicentro, tempo origine, qualità della localizzazione, ecc.

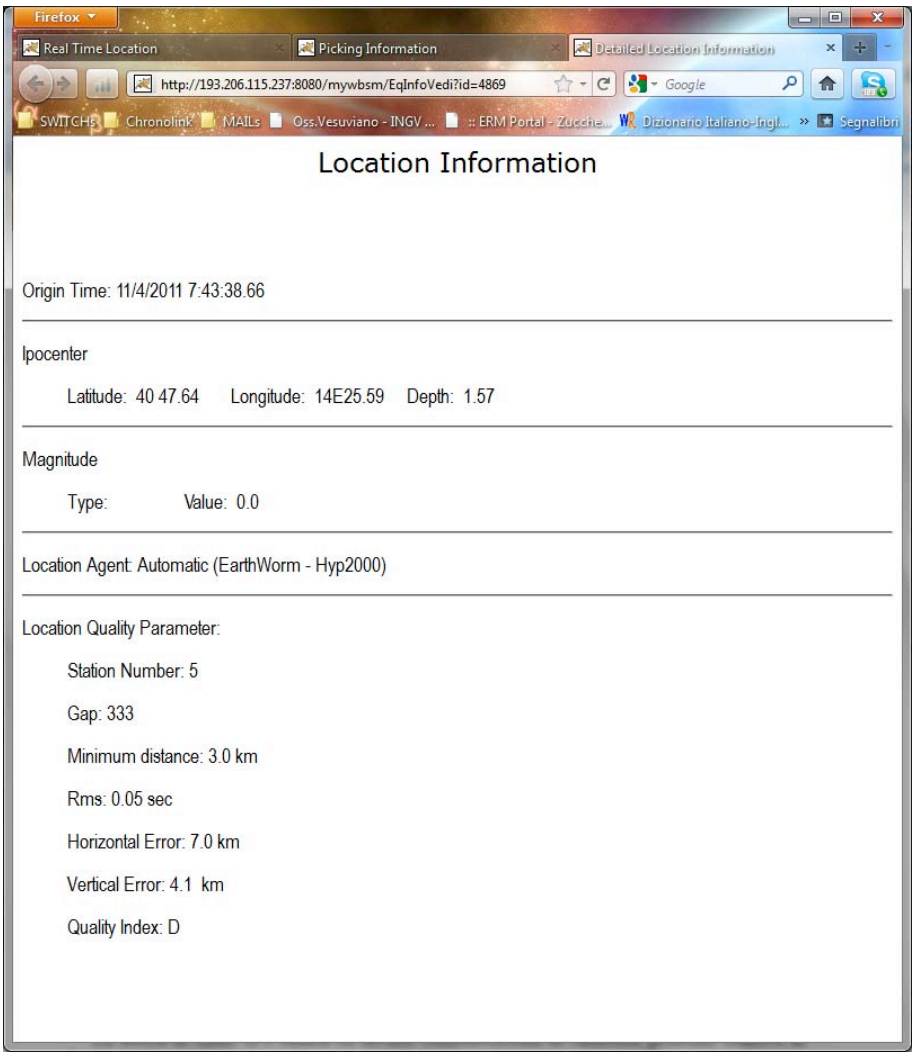


Figura 18. Informazioni estese sulla localizzazione.

Infine nelle figure 19 e 20 è illustrata l'*applet* sviluppata ad hoc per consentire la visualizzazione delle forme d'onda, acquisite dalle stazioni sismiche che hanno contribuito alla localizzazione, potendo effettuare degli zoom ed estrarre graficamente altre informazioni legate alla localizzazione.

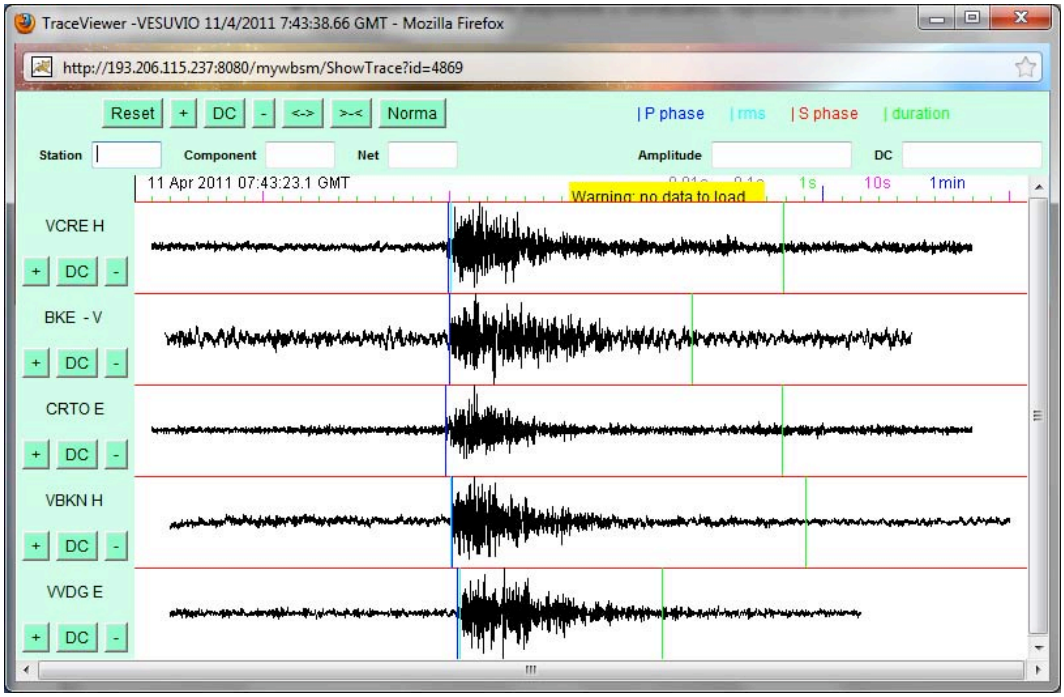


Figura 19. Applet "*Trace Viewer*".

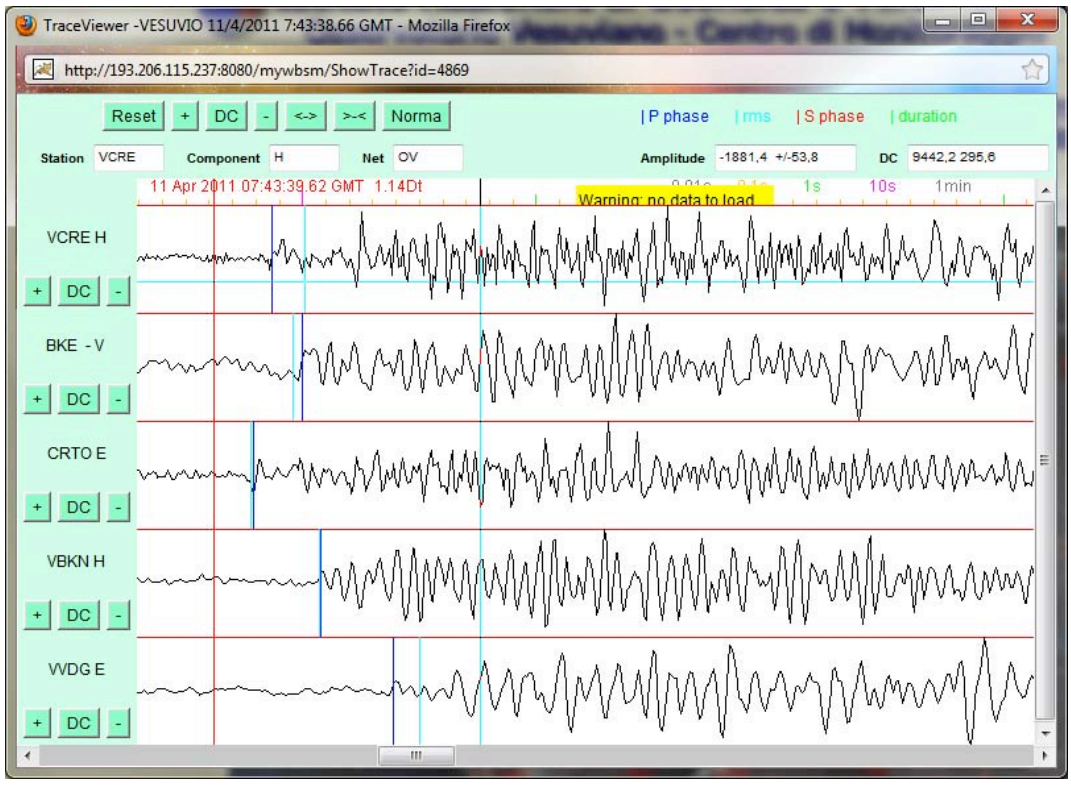


Figura 20. Zoom relativo alle tracce di figura 18.

Ringraziamenti

Si ringrazia Carlo Marcocci per il contributo e i consigli utili al miglioramento del presente lavoro.

Bibliografia

Giudicepietro, F., De Cesare, W., Martini, M. e Meglio, V., (2000). Il Sistema Sismometrico Modulare Integrato (SISMI). Osservatorio Vesuviano Open File Report 6/2002. http://www.ov.ingv.it/pubblicazioni.html#a2000

Giudicepietro, F., Meglio, V., Martini, M. e De Cesare, W., (2002). Web Based Seismological Monitoring (WBSM). Osservatorio Vesuviano Open File Report 3/2002. http://www.ov.ingv.it/pubblicazioni.html#a2002

Harms, D. e Petreley, N., (2001). JSP Servlet e MySQL. Ed. McGrawHill, 492 pp.

Johnson, C.E., Bittenbinder, A., Bogaert, B., Dietz, L. e Kohler, W., (1995). Earthworm: a flexible approach to seismic network processing. IRIS Newsletter, 14 (2), 1-4.

Klein, Fred W., (2002). User's Guide to HYPOINVERSE–2000, a Fortran Program to Solve for Earthquake Locations and Magnitudes. U.S. Geological Survey Open-File Report 02–171.

Lemay, L. e Cadenhead, R., (2003). *Java* 2 SDK 1.4 Guida completa. Ed. APOGEO, 672 pp.

Tottingham, D.M., Lee, W.H.K. e Rogers, J.A., (1989). User manual for MDETECT. In: W.H.K. Lee, Editor, Toolbox for seismic data acquisition, processing, and analysis, IASPEI Software Library Vol., pp. 49–88.

ISSN 2039-7941
Anno **2011**_Numero **195**

appendice

A.1 Struttura XML per lo scambio file

La scelta di XML si è basata su alcune considerazioni di carattere generale relative ai vantaggi di una rappresentazione in tale formato e di carattere specialistico relativamente agli obiettivi di sviluppo di tale lavoro. Il linguaggio XML presenta infatti molti vantaggi:

- possibilità di definire tag user defined;
- rappresentazione del contenuto informativo ad alto livello;
- trasmissione dati tra sistemi eterogenei;
- possibilità di avere associare sulla stessa informazione il formato di visualizzazione su WEB.

In particolare si è scelto di tradurre in documenti XML sia la struttura dati rappresentativa dei parametri di localizzazione, che la struttura dati contenete i *picking*, mentre si è deciso di lasciare in formato SUDS [Tottingham et al., 1989] la struttura dati delle forme d'onda, perché è un formato ampiamente utilizzato in sismologia, supportato da programmi ed utilities diffusi a livello internazionale. La strutturazione del documento XML utilizzato per contenere i parametri di localizzazione è basata sull'analisi dei dati origine in modo da mantenere lo stesso contenuto semantico, ma con una forma di più semplice comprensione. A titolo di esempio di seguito è riportata la stringa che rappresenta la localizzazione nel formato *summary* di *Hypoinverse* in *Earthworm*:

20000916 2020 59.31 40 49.29 14E25.75 0.21 D 1.87 7 83 1. 0.02 0.1 6.4 BW 10786

Questo è un formato di rappresentazione diviso in campi a lunghezza fissa e il contenuto semantico della stringa è del tipo:

- Tempo origine evento:2000/09/16 ore 20:20:59.31
- Latitudine: 40° 49.29' Nord
- Longitudine:14° 25.75' Est
- ecc.…(vedi appendice Formati di *Earthworm*)

e viene tradotto nel seguente documento XML:

```
<?xml version="1.0"?>
<event>
<event_type></event_type>
<time>
<year>2000</year>
<month>9</month>
<day>16</day>
<hour>20</hour>
<min>20</min>
<sec>59.31</sec>
</time>
<hypocenter>
<lat>
<deg>40</deg>
<toward> </toward>
<min>49.29</min>
</lat>
<lon>
<deg>14</deg>
<toward>E</toward>
<min>25.75</min>
```

```
</lon>
<depth>0.21</depth>
</hypocenter>
<magnitude>
<type>D</type>
<value>1.87</value>
</magnitude>
<quality>
<station_number>7</station_number>
<gap>83</gap>
<min_distance>1.0</min_distance>
<rms>0.02</rms>
<err_h>0.1</err_h>
<err_z>6.4</err_z>
<quality_index>B</quality_index>
</quality>
<source>
<type>W</type>
<id_number>10786</id_number>
</source>
</event>
```

Il vantaggio in termini di leggibilità è evidente anche per persone non esperte di sismologia. Con la stessa logica è stato realizzata la traduzione della struttura dati contenente le informazioni di *picking* in un documento XML. Di seguito si riporta un esempio di struttura dati di picking nel formato di *Earthworm* e la stessa struttura tradotta nel formato XML prescelto.

```
33
200010030719464940 4954 14E2555 143 0 5151 1 211482 103 1 2 39199DIS
18 0 39 102 5 0 10 0 0VES WW D 5X 0 0L 0 0 10790D199 10 0 0
$1
BKE NN VV PU0200010030719 4744 0135 0 0 0 0 0 0 29 0 1413500
0 34127 0 0 998 0WD
……..
```

Di conseguenza il formato XML risultante è:

```
<?xml version="1.0"?>
<Pick_list>
<pick>
<id_channel>
<station_name>BKE </station_name>
<network>NN</network>
<component>V </component>
</id_channel>
<p_phase>
<time>
<year>2000</year>
<month>10</month>
<day>3</day>
<hour>7</hour>
<min>19</min>
<sec>47.44</sec>
</time>
<phase_feature>
<on_set> </on_set>
<phase_type>P</phase_type>
```

<polarity>U</polarity>
<picker_weight>0</picker_weight>
</phase_feature>
<rms>0.0</rms>
<delay>0.29</delay>
<other_weight>
<actual_weight>1.35</actual_weight>
<importance>0.998</importance>
</other_weight>
</p_phase>
<s_phase>
<p_post_time>0.0</p_post_time>
<phase_feature>
<on_set> </on_set>
<phase_type> </phase_type>
<picker_weight>0</picker_weight>
</phase_feature>
<rms>0.0</rms>
<delay>0.0</delay>
<other_weight>
<actual_weight>0.0</actual_weight>
<importance>0.0</importance>
</other_weight>
</s_phase>
<duration_magnitude>
<coda_duration>34</coda_duration>
<duration_magnitude_weight>0</duration_magnitude_weight>
<duration_magnitude_value>0.0</duration_magnitude_value>
<formula_label_code>D</formula_label_code>
</duration_magnitude>
<amplitude_magnitude>
<peak_to_peak_amplitude> 0</peak_to_peak_amplitude>
<amplitude_unit>0</amplitude_unit>
<Periodo_di_misura_x_questa_staz.>0.0</Periodo_di_misura_x_questa_staz.>
<amplitude_magnitude_weight>0</amplitude_magnitude_weight>
<amplitude_magnitude_value>0.0</amplitude_magnitude_value>
<formula_label_code> </formula_label_code>
</amplitude_magnitude>
<hypocenter-station_geometric_parameter>
<epicentral_distance>1.4</epicentral_distance>
<azimuth_E_of_N>127</azimuth_E_of_N>
<emergence_angle>135</emergence_angle>
</hypocenter-station_geometric_parameter>
</pick>
…….

Anche in questo caso è evidente la differenza di leggibilità tra i due formati. Nel seguito le strutture dati di localizzazione e di *picking* verranno chiamate rispettivamente *Sum Xml* e *Arc Xml*.

A.2 Formati *Earthworm*

Hypoinverse Archive Format with Shadow Cards

TYPE_HYP2000ARC
(last revised September 8, 1999)

Earthworm has adopted the Y2K-compliant Hypoinverse (aka hyp2000) archive format with shadow cards as Earthworm message TYPE_HYP2000ARC. This is the format in which Earthworm reports earthquake hypocenters and associated phase data. In Earthworm v4.0, TYPE_HYP2000ARC messages are output by eqcoda, read and output by eqverify, hyp2000_mgr
(hyp2000), and read by menlo_report, eqwaves, arc2trig, earlybird, and a host of other processing modules. A TYPE_HYP2000ARC message is a fixed-format message and is built of 6 different types of lines in the following sequence:

1. hypocenter
2. hypocenter shadow
3. phase 1
4. phase shadow 1
5. ...
6. phase N
7. phase shadow N
8. terminator line
9. terminator shadow

The format of each type of line is described in its own section below. Because this format was designed with systems besides Earthworm in mind, some of the fields may remain blank in the messages output by Earthworm. Two sample TYPE_HYP2000ARC messages are included below the format descriptions. Note that this file describes Fortran formats where the column count begins at 1 instead of 0.
STATION ARCHIVE FORMAT YEAR 2000

The following line appears for each station. The unique designation of a station name has expanded with time, reflecting the increase in sensor types and conflicts with station names in use by other
networks. We formally define a station name with 10 letters as the concatenation of the 5-letter site code, 2-letter network code, and 3-letter component code. Hypoinverse optionally uses 1 or 3 letter
component codes, and both are listed in the print file. NCSN practice is to use the 3-letter code for station matching, and the 1-letter code as a convenience label only. The number of decimal places N is implied by the fortran format (ie F5.N).

* indicates an enlarged/moved field
\+ indicates a new field

the weight-out code for P&S times was redundant and has been eliminated.

Start Col.	Len.	Fortran Format	Data
1	5	A5	5-letter station site code, left justified. *
6	2	A2	2-letter seismic network code. *
8	1	1X	Blank *
9	1	A1	One letter station component code.
10	3	A3	3-letter station component code. *
13	1	1X	Blank *
14	2	A2	P remark such as "IP".
16	1	A1	P first motion.
17	1	I1	Assigned P weight code.
18	4	I4	Year. *
22	8	4I2	Month, day, hour and minute.
30	5	F5.2	Second of P arrival.
35	4	F4.2	P travel time residual.
39	3	F3.2	P weight actually used.

42	5	F5.2	Second of S arrival.
47	2	A2	S remark such as "ES".
49	1	1X	Blank
50	1	I1	Assigned S weight code.
51	4	F4.2	S travel time residual.
55	7	F7.2	Amplitude (Peak-to-peak in Develocorder or paper mm). *
62	2	I2	Amp units code. 0=PP mm, 1=0 to peak mm (UCB stations). +
64	3	F3.2	S weight actually used.
67	4	F4.2	P delay time.
71	4	F4.2	S delay time.
75	4	F4.1	Epicentral distance (km).
79	3	F3.0	Emergence angle at source.
82	1	I1	Amplitude magnitude weight code.
83	1	I1	Duration magnitude weight code.
84	3	F3.2	Period at which the amplitude was measured for this station.
87	1	A1	1-letter station remark. (See table 4 below).
88	4	F4.0	Coda duration in seconds.
92	3	F3.0	Azimuth to station in degrees E of N.
95	3	F3.2	Duration magnitude for this station. *
98	3	F3.2	Amplitude magnitude for this station. *
101	4	F4.3	Importance of P arrival.
105	4	F4.3	Importance of S arrival.
109	1	A1	Data source code (See table 1 below).
110	1	A1	Label code for duration magnitude from FC1 or FC2 command.
111	1	A1	Label code for amplitude magnitude from XC1 or XC2 command.

Hypoinverse SUM format

(last revised August 10, 1999)

SUMMARY HEADER FORMAT YEAR 2000

------- ------ ------ ---- ----

* indicates a new/revised field

Start Col.	Len.	Fortran Format	Data
1	4	I4	Year. *
5	8	4I2	Month, day, hour and minute.
13	4	F4.2	Origin time seconds.
17	2	F2.0	Latitude (deg).
19	1	A1	S for south, blank otherwise.
20	4	F4.2	Latitude (min).
24	3	F3.0	Longitude (deg).
27	1	A1	E for east, blank otherwise.
28	4	F4.2	Longitude (min).
32	5	F5.2	Depth (km).
37	3	F3.2	Magnitude from maximum S amplitude from NCSN stations *
40	3	I3	Number of P & S times with final weights greater than 0.1.
43	3	I3	Maximum azimuthal gap, degrees.
46	3	F3.0	Distance to nearest station (km).
49	4	F4.2	RMS travel time residual.
53	3	F3.0	Azimuth of largest principal error (deg E of N).
56	2	F2.0	Dip of largest principal error (deg).
58	4	F4.2	Magnitude of largest principal error (km).
62	3	F3.0	Azimuth of intermediate principal error.
65	2	F2.0	Dip of intermediate principal error.
67	4	F4.2	Magnitude of intermediate principal error (km).

71	3	F3.2	Coda duration magnitude from NCSN stations. *
74	3	A3	Event location remark. (See table 7 below).
77	4	F4.2	Magnitude of smallest principal error (km).
81	2	2A1	Auxiliary remarks (See note below).
83	3	I3	Number of S times with weights greater than 0.1.
86	4	F4.2	Horizontal error (km).
90	4	F4.2	Vertical error (km).
94	3	I3	Number of P first motions. *
97	4	F4.1	Total of NCSN S-amplitude mag weights ~number of readings.*
101	4	F4.1	Total of NCSN duration mag weights ~number of readings. *
105	3	F3.2	Median-absolute-difference of NCSN S-amp magnitudes.
108	3	F3.2	Median-absolute-difference of NCSN duration magnitudes.
111	3	A3	3-letter code of crust and delay model. (See table 8 below).
114	1	A1	Last authority for earthquake N=NCSC (USGS), B=UC Berkeley.
115	1	A1	Most common P & S data source code. (See table 1 below).
116	1	A1	Most common duration data source code. (See cols. 68-69)
117	1	A1	Most common amplitude data source code.
118	1	A1	Coda duration magnitude type code
119	3	I3	Number of valid P & S readings (assigned weight > 0).
122	1	A1	S-amplitude magnitude type code
123	1	A1	"External" magnitude label or type code. Typically "L" (=ML) computed by UCB. This information is not computed by Hypoinverse, but passed along
124	3	F3.2	"External" magnitude.
127	3	F3.1	Total of the "external" magnitude weights (~ number of readings).
130	1	A1	Alternate amplitude magnitude label or type code (i.e. L for ML calculated by Hypoinverse from Wood Anderson amplitudes).
131	3	F3.2	Alternate amplitude magnitude.
134	3	F3.1	Total of the alternate amplitude mag weights ~no. of readings.
137	10	I10	Event identification number
147	1	A1	Preferred magnitude label code chosen from those available.
148	3	F3.2	Preferred magnitude, chosen by the Hypoinverse PRE command.
151	4	F4.1	Total of the preferred mag weights (~ number of readings). *
155	1	A1	Alternate coda duration magnitude label or type code (i.e. Z).
156	3	F3.2	Alternate coda duration magnitude.
159	4	F4.1	Total of the alternate coda duration magnitude weights. *
163	1	A1	Version number of information: 0=25 pick; 1=Final EW with MD; 2=ML added, etc. 0-9, then A-Z. Hypoinv. passes this through.
164	1	A1	Version # of last human review. blank=unreviewed, 1-9, A-Z.

A.3 Piattaforme Supportate da MySQL

Redhat

- Red Hat Enterprise Linux 4
  - o x86
  - o x86_64
  - o Intel IA64
- Red Hat Enterprise Linux 3
  - o x86
  - o x86_64
  - o Intel IA64

Novell

- Novell SuSE Enterprise Linux 9
  - o x86
  - o x86_64
  - o Intel IA64

Debian GNU/Linux

- Debian 3.0
  - o PowerPC
  - o Ultrasparc
- Debian 3.1
  - o PowerPC
  - o x86

Sun Microsytems

- Sun Solaris 10
  - o SPARC, 32/64 bit
  - o X86_64
  - o X86, 32 bit
- Sun Solaris 9
  - o SPARC, 32/64 bit
  - o X86, 32 bit
- Sun Solaris 8
  - o SPARC, 32/64 bit
  - o X86, 32 bit

HP

- HP-UX 11.23
  - o Intel IA64
- HP-UX 11.11
  - o PA-RISC 2.0, 64-bit
  - o PA-RISC 1.1 and 2.0
- HP-UX 11.00
  - o PA-RISC 2.0, 64-bit
  - o PA-RISC 1.1 and 2.0

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