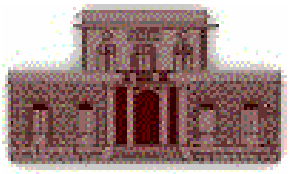
ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA



OSSERVATORIO VESUVIANO

DISPOSITIVO VIRTUALE PER L'ANALISI DA REMOTO DEGLI EVENTI SISMICI (DIVARES)

G.Scarpato **[1]**,***F.Giudicepietro*** **[1]**, ***S.P. Romano*** **[2]**, ***M.Martini*** **[1]**, ***G.Ventre*** **[2]**, ***W. De Cesare*** **[1]**

[1] – INGV-Osservatorio Vesuviano
[2] – Università di Napoli "Federico II" - Dipartimento di Informatica e Sistemistica

Open File Report n°2 - 2004

INTRODUZIONE

I sistemi più avanzati, per il monitoraggio sismico, si avvalgono ampiamente delle moderne tecnologie informatiche. In particolare le tecnologie Web stanno assumendo sempre più un ruolo rilevante come strumento di accesso ai dati e di controllo dell'integrità dei sistemi. In questo contesto è stata sviluppata un'applicazione a supporto delle attività di monitoraggio dei vulcani attivi della Campania, basata sulle tecnologie Web.

I vulcani della Campania (Vesuvio, Campi Flegrei e Ischia), in particolare il Vesuvio e i Campi Flegrei, sono, come è noto, tra quelli a più alto rischio nel mondo a causa dello stile eruttivo, prevalentemente esplosivo, e della presenza nelle loro prossimità di vaste zone urbanizzate. Per sorvegliare queste aree l'Osservatorio Vesuviano (INGV) si serve di reti strumentali che ne consentono il monitoraggio continuo.

Il sistema di monitoraggio sismico è basato su una rete sismica permanente, composta da numerose stazioni distribuite sul territorio, che trasmettono i dati in continuo al centro di acquisizione, via radio e via linea telefonica dedicata. Nella sala di monitoraggio dell'Osservatorio Vesuviano i segnali, invece che sui tradizionali rulli a carta, sono visualizzati su monitor, grazie ad un innovativo sistema di processamento, denominato *SISMI* (vedi Giudicepietro et al. Open file report n°6 2000). Questo sistema ha dei dispositivi di allarme automatico che permettono di ricevere messaggi su telefoni mobili in caso di evento sismico o in caso di malfunzionamento di uno o più dei suoi componenti. Presso la sala di monitoraggio dell'Osservatorio Vesuviano è in uso un sistema software, *Earthworm,* un sviluppato negli Stati Uniti, molto affermato a livello mondiale, che permette di ottenere localizzazioni automatiche degli eventi rilevati dalla rete sismica.

I sistemi automatici per l'invio di messaggi di allarme attualmente sono implementati solo ad uso interno, mentre le comunicazioni agli organi di protezione civile sono effettuate, in modo non automatico, dagli operatori addetti al servizio di sorveglianza, presenti in sede 24 ore su 24. Ciò è dettato dall'esigenza di validare ed, eventualmente, rielaborare i parametri prodotti dai sistemi automatici.

Per questo nasce la necessità di avere a disposizione uno strumento che permetta di effettuare delle analisi e di correggere eventuali errori dei sistemi automatici, operando da remoto. In particolare, è richiesto un dispositivo in grado di fornire ad un addetto alla sorveglianza, o comunque ad un operatore sismologo, la possibilità di visualizzare i segnali rilevati dalle stazioni sismiche e di effettuare le operazioni necessarie all'analisi dei dati, con un qualsiasi computer dotato di un *web browser*. L'esistenza di un ambiente eterogeneo ed interconnesso, ha spinto verso la creazione di applicazioni software distribuite, dove l'informazione è condivisa e la potenza di calcolo distribuita. Tali applicazioni sono in grado di servire contemporaneamente più utenti, garantendo un uso ottimizzato delle risorse di esecuzione. Inoltre l'adozione delle tecnologie, basate sui protocolli Internet (come il TCP/IP, il WEB, etc.) all'interno delle reti locali, ha reso possibile usare il WEB browser come l'interfaccia principe per la consultazione dei dati da remoto, indipendentemente dalla loro natura. A tale scopo, è stato sviluppato un *Dispositivo Virtuale per l'Analisi da Remoto degli Eventi Sismici (DiVARES)*. Esso è basato sul concetto di "applicazioni Web", in modo da dare agli utenti la possibilità di condividere il software e le informazioni in modo semplice ed immediato e favorendo l'accesso anche a gruppi di individui fisicamente lontani.

Nello sviluppo di questo tipo di applicazioni, la comunicazione assume un ruolo fondamentale poiché permette lo scambio dei dati tra i diversi apparati appartenenti all'intera struttura informatica dell'ente: dai sensori più semplici, ai sistemi di controllo più sofisticati, fino ai Pc o Workstation. Inoltre l'adozione di una piattaforma software basata su uno standard comune, sia per quanto riguarda l'interfaccia utente, sia le tecnologie adottate per l'implementazione del software e per la trasmissione dei dati da un'applicazione software all'altra, risulta un requisito ormai fondamentale, soprattutto nei casi in cui si vuole rendere trasparente all'utente l'eterogeneità dei vari apparati che compongono il sistema.

La rapida e crescente diffusione del World Wide Web ha fatto sì che in qualunque PC sia possibile trovare installato una qualche forma di Web browser. Questa situazione può essere favorevolmente sfruttata, per fornire facilmente un front panel del sistema su qualunque macchina che abbia una connessione Internet, senza avere la necessità di scrivere applicazioni client dedicate. In alcuni casi, quindi, l'interfaccia utente di un sistema può essere realizzata sfruttando il browser Web e collegando il sistema stesso ai PC client

attraverso connessioni TCP. L'utente interagisce con il sistema sfruttando applicazioni web sia lato client che lato server per realizzare documenti HTML a contenuto dinamico, che consentano di utilizzare i servizi su un insieme di pagine WEB, da qualunque postazione che sia in grado di eseguire un browser WEB e senza la necessità di acquistare, installare e manutenere software aggiuntivo.

Si è, quindi, scelto di sfruttare applicazioni basate su web per vari motivi:

- **facilità di distribuzione e aggiornamento:** un'applicazione Web si trova interamente sul server, per cui la pubblicazione sul server coincide con la distribuzione e l'aggiornamento effettuato sul server è automaticamente reso disponibile a tutti gli utenti;
- **accesso multipiattaforma:** l'accesso all'applicazione è indipendente dall'hardware e dal sistema operativo utilizzato dagli utenti;
- **riduzione del costo di gestione:** l'uso di Internet come infrastruttura per un'applicazione Web riduce notevolmente sia i costi di connettività sia i costi di gestione dei client (è l'utente stesso che tiene aggiornato e gestisce il proprio "Web browser");
- **scalabilità:** un'applicazione Web ben progettata è in grado di gestire contemporaneamente più utenti in un ambiente basato su server.

I componenti fondamentali di un'applicazione Web sono analoghi per certi versi a quelli di una tradizionale applicazione client/server. Una tipica applicazione client/server è costituita da un client che implementa l'interfaccia utente con alcune funzionalità di elaborazione e di comunicazione e da un server che fornisce una serie di servizi come la gestione e l'accesso ai dati di un database.
Nell'ambito Web l'interazione tra client e server è un po' più articolata per consentire l'integrazione di componenti di varia natura. Un'applicazione Web si basa su elementi software standard indipendenti dalle caratteristiche della particolare applicazione e dalla piattaforma software e hardware su cui è eseguita.
La realizzazione di *DIVARES* è stata articolata su tre livelli logici distinti:

- **Livello dati :** 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, nel caso in esame, delle applicazioni in uso presso l'Osservatorio Vesuviano, quali:

Sistema Sismometrico Modulare Integrato (*SISMI*) e *Earthworm*, che consentono di inviare via rete i segnali provenienti dalle stazioni sismiche verso più centri di raccolta dati, permettono di eliminare una serie di operazioni manuali, spesso banali e ripetitive, che possono comportare ritardi nella comunicazione agli Enti competenti dei fenomeni rilevati e favoriscono la rapida implementazione di nuovi moduli per l'analisi in tempo reale dei dati;

Web Based Seismological Monitoring (*WBSM*), finalizzato alla pubblicazione in tempo reale dei risultati del processing dei segnali sismici ed alla visualizzazione, downloading e analisi dei dati via Internet.

- **Livello intermedio :** tale livello contiene la logica elaborativa dell'applicazione. 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 di 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);

- **Livello di presentazione:** esso costituisce l'interfaccia utente dell'applicazione Web e corrisponde a quello che nelle applicazioni client/server standard è il client. E' costituito da vari componenti combinati tra loro: browser, documenti HTML, applet Java. In figura (a) è mostrata l'interfaccia utente di DIVARES, sviluppata, appunto, con un'applet. La capacità di utilizzo di questi elementi da parte della piattaforma client è uno dei problemi principali nella realizzazione di questo livello. Per questo

l'applicazione è stata testata con la maggior parte di web browser, oggi disponibili. Comunque, le soluzioni da adottare vanno dalla scelta di sfruttare al massimo le capacità elaborative del client, alla realizzazione di un livello di presentazione universale. Un altro elemento da tenere in considerazione è il livello di complessità dell'interfaccia utente dell'applicazione.

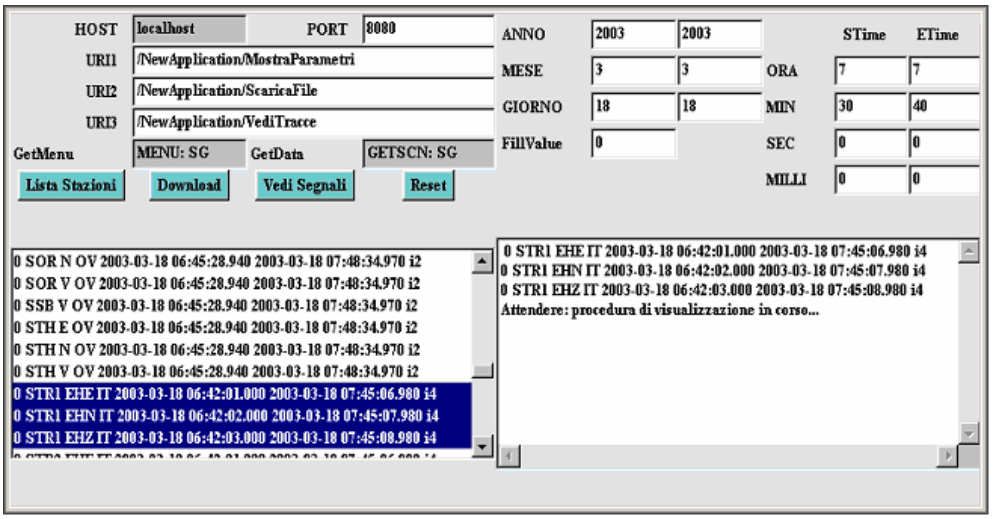


figura (a)

Il linguaggio HTML, nato per la distribuzione di documenti in ambienti distribuiti, non consente di progettare interfacce utenti molto avanzate. Una soluzione è quella di prevedere un insieme di pagine HTML standard, arricchite da applet Java.

Nei successivi capitoli saranno analizzati in maniera più dettagliata i particolari implementativi e le specifiche di progetto che hanno determinato l'adozione di specifiche scelte. In particolare, allo scopo di modellare tale sistema, è stato utilizzato *Unified Modelling Language* (UML) come linguaggio di modellazione del software orientato agli oggetti, grazie al quale è possibile realizzare una documentazione esaustiva dei sistemi software, ma anche, in fase di progettazione, definire in maniera chiara le specifiche di progetto, gli oggetti da creare, il ruolo che essi giocano nel complessivo sistema e la loro reciproca interazione.

Capitolo 1

MONITORAGGIO SISMICO DEI VULCANI DELLA CAMPANIA

Per effettuare il monitoraggio sismologico delle aree vulcaniche della Campania, l'Osservatorio Vesuviano ha sviluppato e mantiene una rete sismica regionale che trasmette i dati in continuo al centro di sorveglianza. La configurazione attuale della rete comprende 33 stazioni analogiche a corto periodo (1Hz) e 3 stazioni digitali a larga banda.

La figura 1.1 mostra la distribuzione delle stazioni sul territorio.

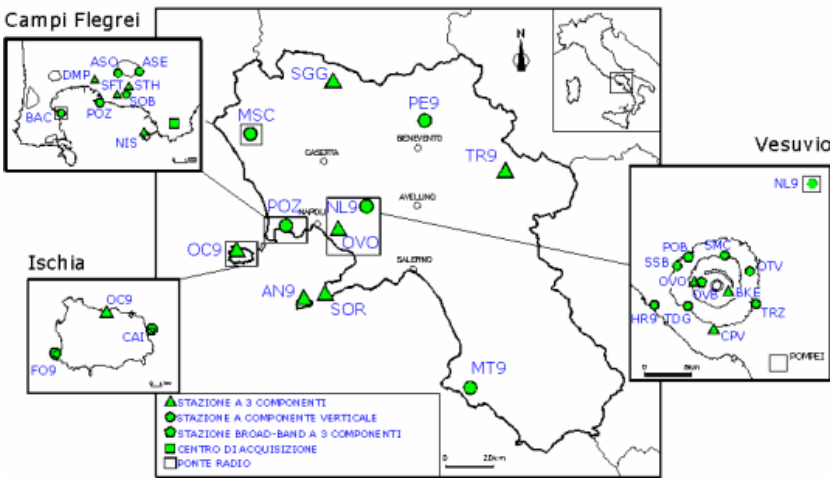


figura 1.1

Come si può notare le stazioni sono disposte in configurazione densa nelle aree vulcaniche con maggiore attività sismica (Vesuvio e Campi Flegrei).

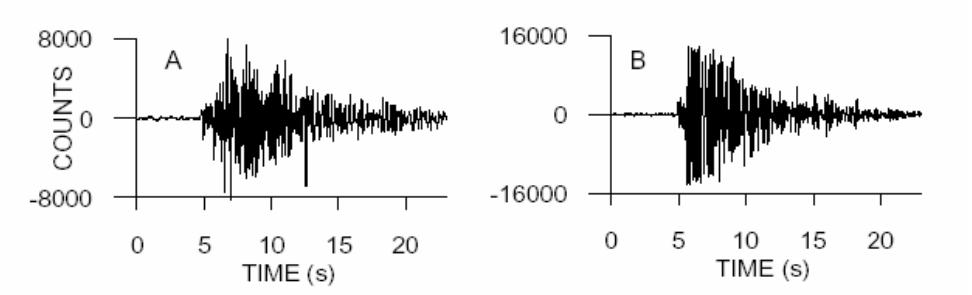
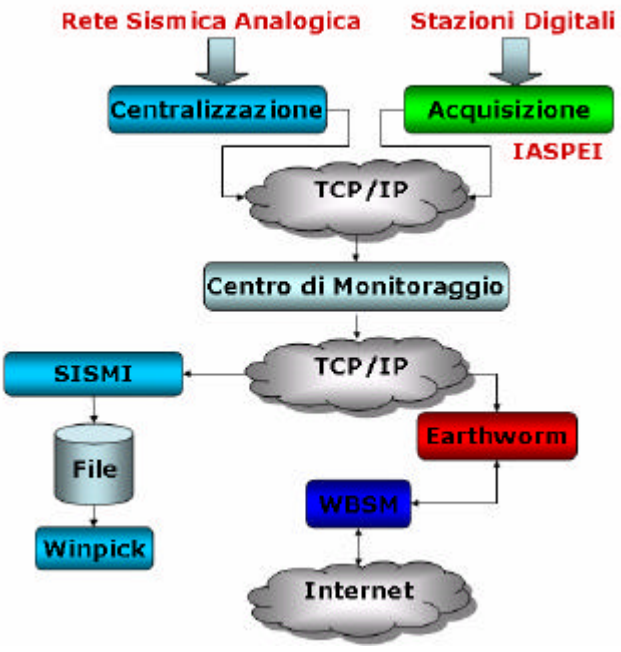


figura 1.2

Questa sismicità è molto locale e produce generalmente terremoti di piccola magnitudo solo raramente avvertibili dalle popolazioni residenti. Esempi del sismogramma di un terremoto locale del Vesuvio(A) e di uno dei Campi Flegrei(B) sono mostrati nella figura 1.2. Le stazioni della Rete Sismica dell'Osservatorio Vesuviano (RSOV) trasmettono i segnali via radio o con linee telefoniche dedicate al Centro di Monitoraggio dove questi sono processati dal Sistema Sismometrico Modulare Integrato (SISMI) e, successivamente, controllati ed archiviati dagli analisti del laboratorio sismico dell'Osservatorio Vesuviano. Allo stato attuale l'impegno delle istituzioni e dei gruppi di lavoro coinvolti nell'implementazione e gestione dei sistemi di monitoraggio sismologico è fortemente orientato allo sviluppo di sistemi automatici per il processamento dei dati. I sistemi di nuova generazione sono progettati per realizzare prestazioni più avanzate, soprattutto riguardo gli aspetti dell'analisi in tempo reale, della trasmissione dati e del controllo remoto dei processi. Tra questi citiamo **Earthworm,** un sistema *open source* sviluppato presso la *United States Geological Survey (USGS)*, e **SISMI**, sviluppato presso l'Osservatorio Vesuviano, che sono utilizzati congiuntamente presso il Centro di Monitoraggio - OV.

In figura è mostrata schematicamente l'interazione tra i vari sistemi di acquisizione ed analisi installati presso il Centro di Monitoraggio dell'Osservatorio Vesuviano.



In blue sono riportati i sistemi progettati e presso l'Osservatorio Vesuviano.

In verde è riportato il sistema IASPEI, sviluppato presso l'USGS, più tradizionale rispetto ad Earthworm ed a SISMI, ed in rosso è rappresentato Earthworm.

I vantaggi offerti da Earthworm, SISMI e WBSM sono molteplici. Essi consentono di inviare via rete i segnali provenienti dalle stazioni sismiche verso più centri di raccolta dati,

permettono di eliminare una serie di operazioni manuali, spesso banali e ripetitive, che possono comportare ritardi nella comunicazione dei fenomeni rilevati agli Enti competenti e favoriscono la rapida implementazione di nuovi moduli per l'analisi in tempo reale dei dati. Quest'ultimo 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, può essere implementata in maniera automatica ed applicata in tempo reale grazie alle caratteristiche dei moderni sistemi di processamento dei dati. Ciò qualifica e rende più efficace l'attività di monitoraggio consentendo di interpretare meglio i fenomeni che si rilevano.

1.1 Sistema Sismometrico Modulare Integrato(SISMI)

SISMI è un sistema distribuito per l'acquisizione e la gestione di dati sismici sviluppato presso il Centro di Monitoraggio dell'Osservatorio Vesuviano ([9]-Giudicepietro et al.,2000). La configurazione attualmente in uso all'Osservatorio Vesuviano comprende più di 20 moduli che comunicano mediante messaggi di rete TCP/IP ed UDP e sono distribuiti su 12 PC e 31 schermi dislocati in due diverse sedi.

SISMI è utilizzato per:

- Acquisizione dei segnali delle stazioni analogiche;
- Trasmissione dei dati da un punto di centralizzazione al Centro di Monitoraggio via rete;
- Archiviazione dei dati su disco;
- Visualizzazione in continuo dei segnali su monitor;
- Inserimento nel sito web dei segnali sismici in tempo reale ;
- Stampa giornaliera dei segnali relativi ai canali più significativi;
- Analisi in continuo delle proprietà spettrali e di polarizzazione del rumore sismico;
- Detezione degli eventi;

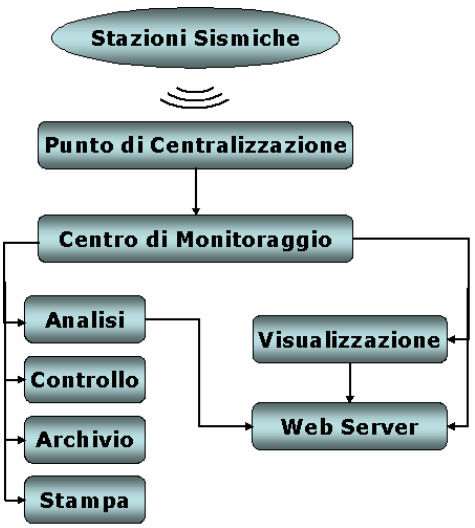


figura 1.3

- Notifica degli eventi;
- Interfaccia con il sistema IASPEI;
- Controllo dell'integrità del sistema.

La figura 1.3 mostra uno schema semplificato dell'architettura di SISMI.

1.2 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 opportuno protocollo.

I moduli di Earthworm scambiano dati e messaggi utilizzando un sistema di *broadcasting* basato su *buffer* circolari di memoria condivisa detti *ring*.

Earthworm è utilizzato presso l'Osservatorio Vesuviano per:

- *Picking* automatico delle fasi;
- Localizzazione automatica degli eventi;
- Accesso ai dati in tempo reale da remoto;
- Archiviazione temporanea dei dati su disco;
- Interfaccia con WBSM;
- Scambio dati con le altre Sezioni dell'Istituto Nazionale di Geofisica e Vulcanologia (es. CNT di Roma e di Catania).

1.3 WBSM

WBSM è un sistema modulare, sviluppato in collaborazione con il Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II, in cui la funzione di interfaccia verso le fonti di dati è affidata a moduli specifici. Perciò se si desidera fare interagire questo sistema con una qualsiasi fonte di dati, è sufficiente effettuare la sostituzione o la modifica dei soli moduli dedicati all'interfaccia ed alla omogeneizzazione dei formati. Attualmente sono stati presi in considerazione come fonti di dati Earthworm ed il programma di analisi Winpick, sviluppato presso l'Osservatorio Vesuviano (INGV) . Questi sono i sistemi utilizzati presso il Centro di Monitoraggio dell'Osservatorio Vesuviano per la localizzazione degli eventi sismici. Earthworm è utilizzato per le localizzazioni automatiche ed interagisce con il Sistema Sismometrico Modulare

Integrato (SISMI) che gli fornisce i dati. Winpick è il programma utilizzato dagli operatori per l'analisi manuale e la localizzazione degli eventi. L'interfaccia con Earthworm consente di costituire un archivio automatico di eventi, non supervisionato, e di pubblicare immediatamente i risultati delle localizzazioni senza alcun intervento da parte degli operatori. Nel caso di reti sismiche locali con un numero di stazioni contenuto, come sono spesso le reti per il monitoraggio sismologico dei vulcani, il criterio di associazione dei *picking* in un evento, a volte, può fallire e, quindi, portare a false localizzazioni. Per questo motivo WBSM, fornisce non solo i parametri ipocentrali, ma anche tutti i parametri calcolati dal programma di localizzazione ed inoltre ha un *viewer* per visualizzare le forme d'onda, mettendo così a disposizione tutti gli strumenti necessari per il riconoscimento degli eventi spuri. L'interfaccia verso Winpick consente, invece, la costituzione di un archivio supervisionato e può assolvere alla funzione di comunicazione al pubblico e di distribuzione dei dati. WBSM è attivo, in fase di test, presso il Centro di Monitoraggio dell'Osservatorio Vesuviano dall'inizio del 2001, interfacciato con Earthworm.

In sintesi WBSM offre i seguenti servizi:

- Pubblicazione delle informazioni relative alle localizzazioni automatiche e di quelle effettuate dagli operatori
- Costituzione automatica di una base dati con forme d'onda e dati parametrici
- Interfaccia Web per le localizzazioni automatiche
- Accesso ai dati da remoto
- Analisi delle forme d'onda da remoto
- Interfaccia Web per i sistemi di analisi off line
- Localizzazione degli eventi con procedura *off line*
- Archiviazione su disco rigido
- Interfaccia con i sistemi Web

Con il sistema DIVARES, si intende fornire l'operatore sismologo della possibilità di interagire con specifici moduli di Earthworm, per la visualizzazione delle forme d'onda di una qualunque stazione sismica, funzionante nella RSOV. In questo modo è possibile effettuare un'analisi dei sismogrammi, estrarre i parametri di picking e inviarli ad opportuni moduli integrati in Earthworm. In questo modo è possibile retroazionare ad Earthworm i parametri estratti durante l'operazione di picking, condizionando Earthworm a localizzare un evento sismico in base ai parametri forniti dall'operatore, piuttosto che a quelli

generati dalla completa analisi automatica. Di conseguenza vengono innescate tutte le procedure caratterizzanti il funzionamento del sistema Earthworm-WBSM.

Quindi DIVARES permette l'analisi dei sismogrammi e la localizzazione degli eventi da remoto, consentendo l'attività di sorveglianza anche da postazioni non residenti presso l'Osservatorio Vesuviano.

Nel prossimo capitolo verrà descritta in dettaglio l'architettura di Earthworm e le modalità di interfacciamento con questo, necessarie per la realizzazione della nuova applicazione. Nei capitoli successivi saranno descritte le specifiche di progetto e i dettagli implementativi.

Capitolo 2

Livello dati

Negli ultimi decenni le tecniche ed i sistemi di monitoraggio sismologico hanno avuto una rapida evoluzione, legata al generale progresso che ha consentito lo sviluppo di strumentazione sempre più sofisticata e reso disponibili sistemi di calcolo ad elevate prestazioni, relativamente a basso costo. Questi strumenti, supportati da una capillare rete informatica, hanno aperto nuove prospettive per la trasmissione dei dati, per la loro elaborazione in tempo reale e per il controllo remoto dei processi.

I sistemi di processamento dei dati sismici adottati presso l'Osservatorio Vesuviano sono:

- Earthworm, un sistema *open source*, sviluppato presso l'United States Geological Survey (USGS);
- SISMI, sviluppato presso il Centro di Monitoraggio dell'Osservatorio Vesuviano (Giudicepietro et al., 2000);
- Web Based Seismological Monitoring (WBSM), sviluppato presso il Centro di Monitoraggio dell'Osservatorio Vesuviano.

Questi sistemi, opportunamente integrati tra loro, permettono l'acquisizione dei segnali sismici trasmessi in continuo dalle stazioni della RSOV e la localizzazione automatica degli eventi.

Nel caso di reti sismiche locali con un numero di stazioni contenuto, come sono spesso le reti per il monitoraggio sismologico dei vulcani, il criterio di associazione dei "picking" in un evento, utilizzato dai sistemi automatici, può fallire e portare a false localizzazioni. Il presente lavoro nasce dall'obiettivo di fornire uno strumento di analisi *off line* dei sismogrammi che consenta di correggere le localizzazioni automatiche e di effettuare nuove localizzazioni a partire dal rilevamento manuale dei parametri effettuato da remoto.

In figura 2.1 è illustrata l'architettura complessiva del sistema e come questo si interfaccia con i preesistenti sistemi.

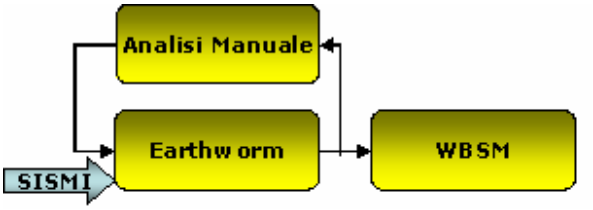


figura 2.1

Dalla figura si possono notare le funzionalità che il sistema complessivo offre.

Il ruolo principale è giocato da Sismi ed Earthworm, il primo per l'acquisizione dei dati sismici rilevati dalle stazioni della RSOV, il secondo per la localizzazione automatica degli eventi.

Il blocco WBSM è finalizzato alla pubblicazione in tempo reale dei risultati del processamento dei segnali sismici ed alla visualizzazione, *downloading* e analisi dei dati via Internet, fornendo tutti i parametri di qualità calcolati da Earthworm.

Il blocco di analisi manuale può intervenire quando il criterio di associazione dei *picking* fallisce ed è necessario che l'operatore sismologo ne effettui una correzione. In tal caso, è possibile visualizzare i sismogrammi utili, rilevare manualmente tutte le informazioni di *picking* necessarie alla localizzazione ed inviarle ad un modulo servente. Questo ha il compito di formattare i dati che riceve in ingresso in un formato che sia compatibile con eventuali applicazioni che effettuano analisi automatiche, come ad esempio Earthworm per la localizzazione automatica dell'evento sismico.

Nei paragrafi che seguono viene data una descrizione più dettagliata di Earthworm e WBSM, utilizzati come strumenti di sviluppo per il lavoro di tesi, presentato nei capitoli 3 e 4.

2.1 Earthworm: i dettagli

Earthworm, come già accennato nel capitolo 1, è un sistema modulare per il processamento automatico dei dati sismici, che prevede degli strumenti propri per l'acquisizione dati.

Tuttavia è possibile immettere e/o prelevare dal sistema informazioni via rete, interagendo con gli opportuni moduli, tramite un appropriato protocollo di comunicazione. Questa possibilità permette, da un lato, di estrarre informazioni sui segnali acquisiti in tempo reale, dall'altro di inserire informazioni per la correzione manuale di eventuali mancate o errate localizzazioni.

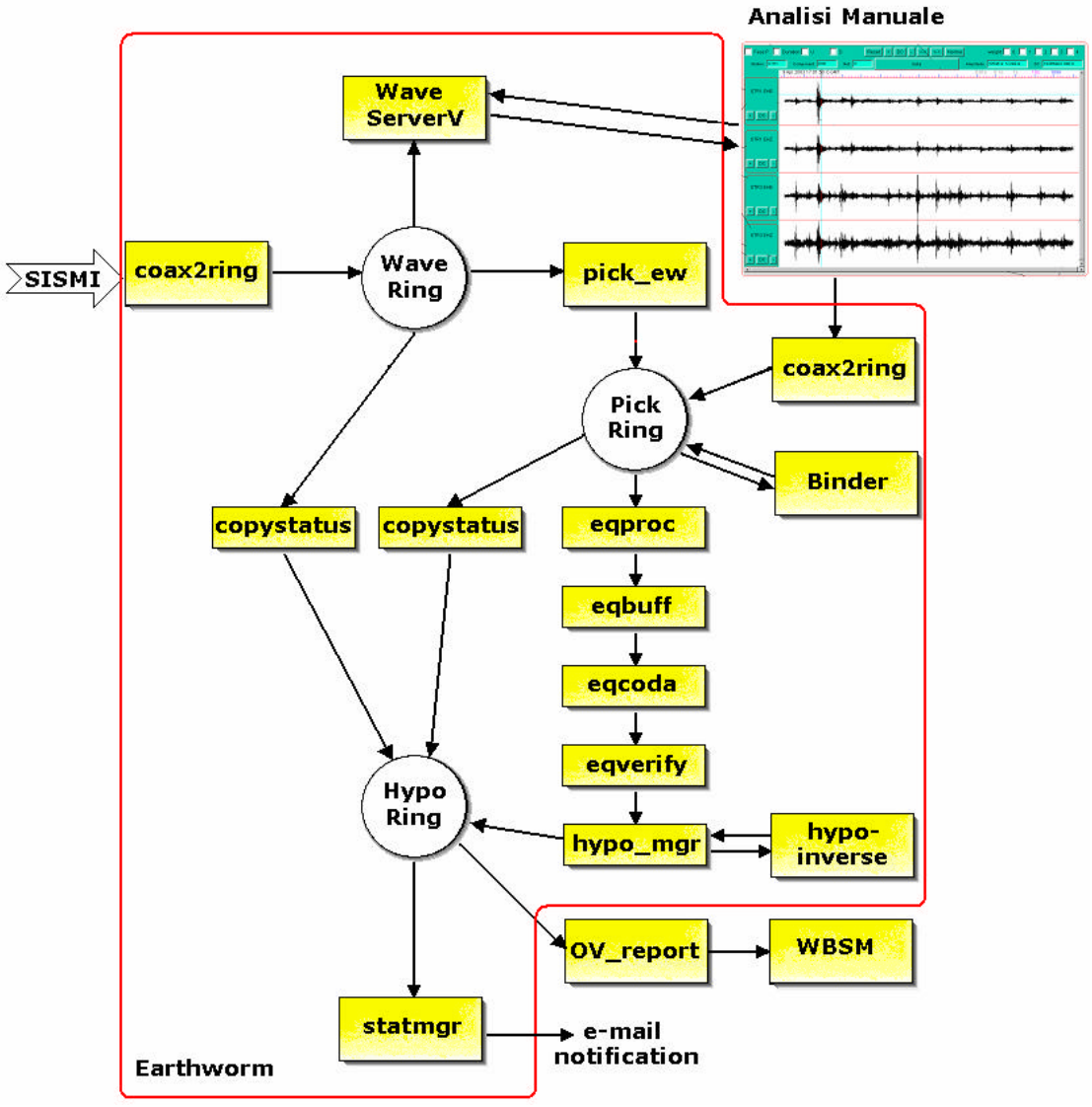


figura 2.2

In figura 2.2 è illustrata l'architettura complessiva del sistema, con evidenziati i moduli principali di Earthworm.

Come si può notare, la struttura di base di Earthworm, così come configurato al centro di Monitoraggio dell'Osservatorio Vesuviano, è dotata di tre *buffer* circolari, detti "ring", su cui sono in ascolto i moduli del sistema.

In particolare, sul "Wave Ring" sono in ascolto tre moduli:

- Wave ServerV, che effettua il *buffering* temporaneo delle forma d'onda acquisite per ogni specifico canale;
- Pick_ew, che rileva eventuali fasi sismiche ed invia i picking sul "Pick Ring";
- Copystatus che semplicemente copia gli eventuali massaggi di errore su "Hypo Ring".

Sul *Pick Ring*, sono in ascolto i seguenti moduli:

- Binder_ew che associa i picking in eventi sismici. Questo modulo ascolta i messaggi dal pick_ew e produce localizzazioni di massima;
- Eqproc che è il primo di una serie di sottomoduli che effettuano la localizzazione dell'evento sismico. I sottomoduli sono collegati tra loro tramite delle pipe e sono visti da Earthworm come un unico modulo. Eqproc assembla la localizzazione di massima del binder_ew con le relative fasi prodotte da pick_ew e manda l'evento così assemblato ad eqbuff. Quest'ultimo gestisce una coda di eventi verso il successivo processo di elaborazione, eqcoda. Questo processo calcola la durata degli eventi. Infine eqverify effettua una serie di controlli per eliminare eventi spuri e hypo_mgr effettua la localizzazione finale dell'evento per poi inviarla sull'HypoRing.

Infine, dalla figura 2.2 si può notare la presenza di un ulteriore modulo in ascolto sull'Hypo Ring:

- OV_report, che è il modulo sviluppato per interfacciare WBSM con Earthworm. Questo modulo è utilizzato per spedire l'evento sismico rilevato, completo delle informazioni di localizzazione, picking e delle forme d'onda, al sistema remoto di archiviazione/elaborazione di WBSM.

I dati sono immessi nel sistema tramite il modulo coax2ring. Questo è un modulo servente che "ascolta la rete" in attesa di eventuali messaggi da inserire in Earthworm. Il sistema è stato configurato in modo da avere due di questi moduli attivi, uno per importare i segnali sismici che, via rete, provengono da SISMI in tempo reale, l'altro per inserire le informazioni di picking nel Pick Ring. E' proprio grazie a quest'ultimo modulo che, tramite

connessione socket su protocollo UDP, è possibile immettere nel sistema di localizzazione automatica i parametri di picking rilevati con procedure manuali.
Nel prossimo paragrafo viene illustrato in dettaglio il modulo Wave ServerV, utilizzato come fonte dati per l'estrazione delle forme d'onda.

2.1.1 Wave ServerV

Il Wave ServerV [8] è un server sincrono, nel senso che, ricevuta una richiesta, tutte le operazioni di elaborazione vengono portate a termine prima di elaborarne un'altra. Poiché ci possono essere diversi *thread* del server, più di un client può connettersi e effettuare richieste di dati. A tale scopo nell'*header* di una richiesta è previsto un campo, "request id", in modo che il Wave ServerV possa assistere "clienti asincroni". Infatti, così facendo, il codice di processing delle richieste risulta essere strettamente collegato al particolare codice cliente e ci si deve solo preoccupare di ricordare quale replica associare alla particolare richiesta. Tale identificativo, dovendo essere presente nell'header della richiesta da inviare, viene generato dal client ed è replicato dal server nella risposta. Riguardo alla modalità di comunicazione tra client e server, questa avviene con protocollo TCP/IP.
In definitiva, il Wave ServerV acquisisce le tracce di specifici canali e le mantiene in una coda circolare, la cui dimensione può essere configurabile ad hoc, ed è capace di fornire, su richiesta, specifiche porzioni di segnale per ogni canale.
Gli aspetti base del modulo includono:

- La gestione di al più 10 *client* concorrenti;
- Dimensione del *buffer* configurabile dall'utente-gestore, fino a 2 gigabyte per traccia (approssimativamente 100 giorni, se si assume un *sampling rate* di 100 campioni/s e rappresentazione dei dati a 16 bit);
- La rappresentazione dei dati delle tracce è o di tipo ASCII, utile ad esempio a scopo di visualizzazione su display, o binario; tali dati contengono tutte le informazioni così come ricevute dalle stazioni sismiche.

Le richieste possono essere di due tipi, MENU e GET. La prima consente di estrarre dal server informazioni sulle stazioni presenti nella lista delle stazioni attive, la seconda di estrarre le forme d'onda di tutte le stazioni, che si desidera analizzare. In particolare, una richiesta delle stazioni attive può avvenire secondo tre modalità:

1. MENU:*<request id>*;
2. MENUSCN:*<request id><s><c><n>*;
3. MENUPIN:*<request id><pin#>*,

dove la terna S-C-N sta per *Station-Component-Network.*

Nel primo caso, si riceverà una risposta del tipo:

<request id>pin#<s><c><n><startTime><endTime><datatype>…↵
…pin#<s><c><n><startTime><endTime><datatype>\n

ricevendo in questo modo la lista dei canali che vengono acquisiti e l'intervallo di tempo valido per ogni canale.

Negli altri due casi, si riceverà una risposta del tipo:

<request id><pin#><s><c><n><datatype><\n>

in modo da ottenere informazioni su un canale in base ad una specifica terna S-C-N, o in base al terminale "fisico" della scheda di acquisizione dati associato a quel canale.

Per quanto riguarda le richieste di tipo GET, queste possono essere inoltrate secondo due modalità:

1. GETSCN: <request id><s><c><n><startTime><endTime><fill-value>;
2. GETPIN: <requestid><pin#><startTime><endTime><fill-value>.

Nel primo caso, si otterrà una risposta del tipo:

<requestid><pin#><s><c><n>F<startTime><endTime><samplingRate>↵
sample(1)sample(2)…sample(N)<\n>

dove i campioni ottenuti, appartenenti all'intervallo di tempo specificato da "endTime-startTime", sono in formato ASCII e F è uno di un gruppo di flag di controllo. I particolari sono chiariti in tabella 2.1.

Flag	Descrizione
FR	Tutti i dati richiesti sono a destra del tank
FL	Tutti i dati richiesti sono a sinistra del tank
FG	Tutti i dati richiesti sono in un gap particolare del tank
FB	La richiesta del client non è contemplata
FN	I canali richiesti (tank) non sono stati trovati in questo Wave Server
FU	Si è verificato un errore non noto

tabella 2.1

Inoltre, tutte le porzioni di dati all'interno del gap richiesto, che presentano dei "buchi" a causa di errori di trasmissione o acquisizione, vengono riempiti con un valore impostato dal client: fill-value.
Nel secondo caso, analogamente al primo, si ottengono i dati delle tracce, ma, in questo caso, relative ad uno specifico numero di pin della scheda di acquisizione dati.

2.2 WBSM

WBSM ha lo scopo di interfacciare Earthworm al Web, realizzando un sistema di archiviazione ed elaborazione degli eventi sismici che consenta l'accesso, la visualizzazione, il downloading e l'analisi dei dati, prodotti da Earthworm, via Internet.
Le funzionalità richieste da WBSM sono distribuite in tre sottosistemi:

- Il sistema di gestione degli eventi, che ha il compito di captare un evento sismico da una fonte di dati, convertire in XML parte delle informazioni e spedire l'evento ad un centro di archiviazione;
- Il sistema di gestione dei documenti, che svolge il ruolo di archiviazione ed elaborazione degli eventi sismici;
- L'interfaccia End User, che ha il compito di presentare all'utente gli eventi sismici disponibili presso il centro remoto di archiviazione-elaborazione.

In WBSM un evento sismico è rappresentato da tre strutture dati:

- Una struttura dati contenente i parametri della localizzazione epicentrale;
- Una struttura dati contenente le informazioni relative al picking delle fasi ed altri parametri calcolati dal programma di localizzazione, utili alla valutazione della qualità della localizzazione;
- Una struttura dati contenente le forma d'onda dell'evento sismico.

In sintesi, quando avviene un evento sismico, i moduli di Earthworm lo rilevano e spediscono le strutture dati di localizzazione e di picking sull'apposito ring e, parallelamente, scrivono su un file la struttura dati delle forma d'onda. Il sistema di gestione degli eventi sismici capta dagli appositi ring le strutture dati di localizzazione e di picking e va a recuperare il file contenente le forme d'onda. Il sistema di gestione degli eventi trasforma, poi, le strutture dati della localizzazione e del picking in formato XML e le trasmette, insieme al file delle forme d'onda, al sistema di gestione dei documenti. Questo componente, che risiede su un Web server, offre una serie di servizi per l'archiviazione e la gestione dei documenti ricevuti. Alcuni di questi servizi sono utilizzati da applicazioni lato server del sistema di interfaccia end-user, che realizza il rendering dei dati. L'interfaccia end-user comprende anche applicazioni lato client che permettono la visualizzazione grafica dei dati e la loro analisi.

Tale interfaccia permette all'utente la visualizzazione, la ricerca e l'analisi degli eventi sismici disponibili su una macchina remota. In particolare si suppone che la macchina remota metta a disposizione, oltre al servizio di accesso ai dati, anche il servizio di interrogazione dei dati immagazzinati, necessario nelle fasi di ricerca ed analisi degli eventi sismici. L'interfaccia è realizzata tramite un sistema distribuito, composto da due classi di moduli:

- I moduli server, residenti sulla macchina remota;
- I moduli client, residenti sulla macchina utente.

Ai moduli client sono demandate tutte le operazioni di interazione grafica con l'utente che non richiedono un accesso ai dati, mentre ai moduli server sono affidati i compiti di reperimento delle informazioni che andranno visualizzate nella pagina Web richiesta.

In figura 2.3 è mostrata la pagina web che rappresenta l'interfaccia utente verso WBSM.

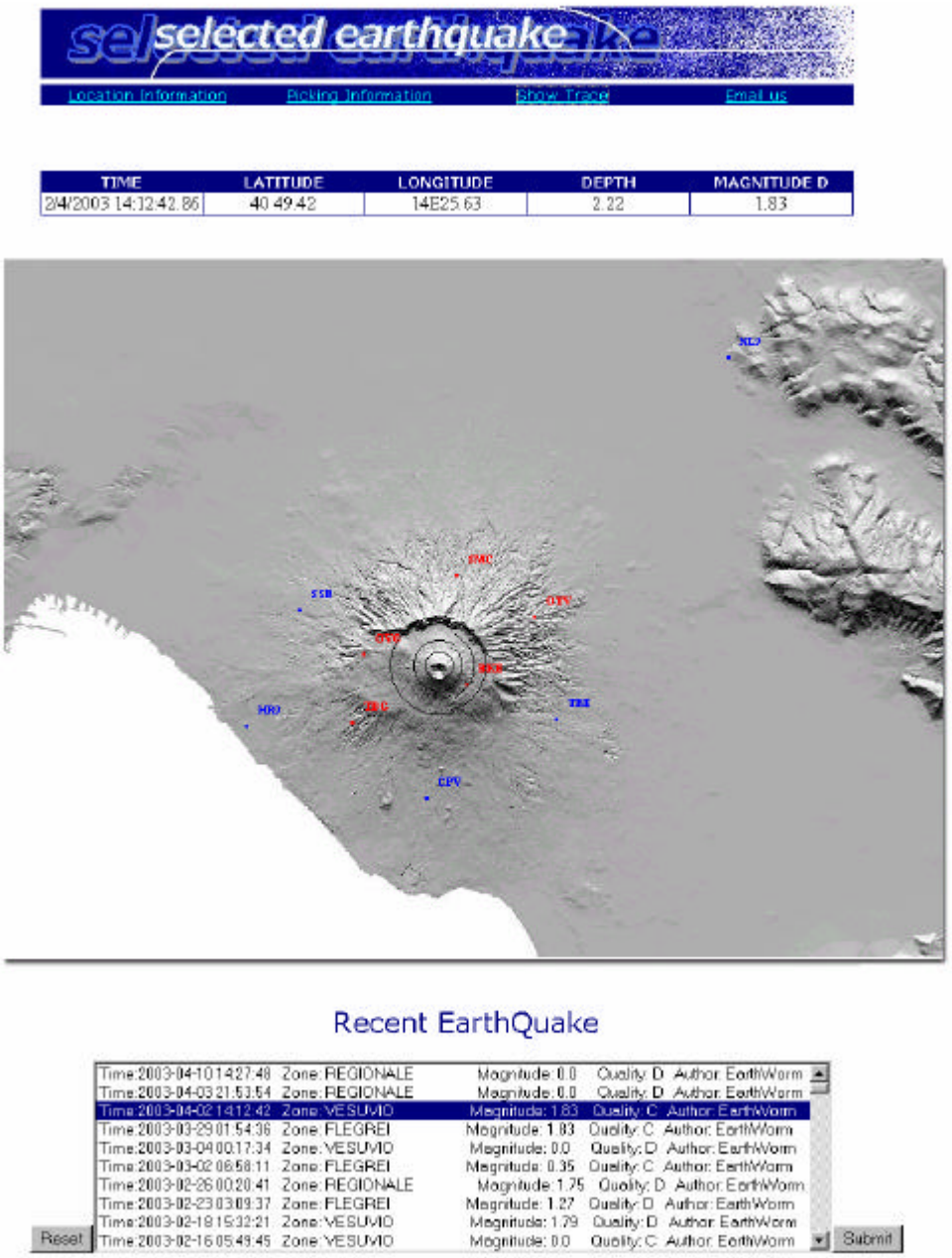


figura 2.3

Queste pagine hanno la funzione di:

- presentare l'ultimo terremoto avvenuto su una mappa della zona dell'epicentro;
- visualizzare i suoi parametri fondamentali di localizzazione in una apposita tabella;

- visualizzare le stazioni sismiche dislocate sul territorio ed evidenziare quelle che hanno paretcipato attivamente alla localizzazione dell'evento;
- permettere la selezione e la visualizzazione degli ultimi 150 terremoti avvenuti, attraverso un apposito elenco;

Tra questi eventi sismici ci saranno quelli che Earthworm ha localizzato in base alle procedure automatiche e quelli che, invece, ha localizzato sulla base delle informazioni che l'utente ha fornito tramite la nuova applicazione, sviluppata nell'ambito del presente lavoro e i cui dettagli implementativi verranno discussi nei prossimi capitoli.

Capitolo 3

Livello intermedio: le servlet

Le Servlet sono moduli software scritti in Java che vengono eseguiti in applicazioni lato server per esaudire le richieste dei client. Esse non sono legate ad un particolare protocollo per la comunicazione tra client e server, anche se più comunemente si utilizza il protocollo HTTP.

Per scrivere una Servlet si fa uso delle classi del package javax.servlet che è il framework di base per la scrittura delle servlet e del package javax.servlet.http che è l'estensione del framework di base, per la comunicazione via http. Utilizzando un linguaggio portabile come Java, le Servlet consentono di realizzare soluzioni per estendere le funzionalità dei server web, indipendenti dal sistema operativo su cui esse vengono eseguite. Rispetto alle tradizionali applicazioni lato server, le Servlet hanno diversi vantaggi. Più precisamente, esse presentano i seguenti vantaggi:

- *Efficienza*. Con una tradizionale applicazione CGI, viene generato un processo (istanza del programma o script CGI) per ogni richiesta che arriva al server, questo tipo di operazione può risultare abbastanza pesante in termini di risorse. Con le Servlet invece, la JVM (Java Virtual Machine) genera, per ogni richiesta da esaudire, un Thread Java, molto più leggero di un processo generato dal sistema operativo: con il CGI se ci sono N richieste contemporanee, esistono N immagini in memoria del programma CGI; con le Servlet invece, ci sono N Thread ma una sola copia della classe Servlet in memoria. Inoltre c'è da dire che l'efficienza è ancora superiore se si pensa che, ad esempio, un'applicazione CGI che si interfaccia con un database si connette ad esso ad ogni richiesta che arriva e si disconnette al termine: operazioni queste molto pesanti. Con le Servlet invece è possibile mantenere aperte una o più connessioni al database, utilizzando ad esempio un Connection Pool, e riutilizzarle per richieste differenti.
- *Facilità d'uso.* Java mette a disposizione una API per la scrittura delle servlet che facilità molto le varie operazioni coinvolte nell'utilizzo di una Servlet e cioè: manipolazione dei dati provenienti dai form HTML, lettura e gestione delle intestazioni (headers) delle richieste http, gestione dei Cookie e delle sessioni e molte altre cose utili.

- *Potenza*. Con le servlet si possono fare tante cose che con i programmi CGI sono difficili da realizzare: le servlet possono dialogare direttamente con il server web facilitando l'utilizzo e la condivisione di dati (testi, immagini ecc.) . Inoltre le servlet facilitano operazioni come la session tracking, permettendo il mantenimento di informazioni tra una richiesta e un'altra, e di caching delle operazioni già eseguite in una richiesta precedente.
- *Portabilità* . Essendo scritte in Java mediante un'API standard, le Servlet possono girare su qualsiasi piattaforma. Oggi le servlet sono supportate, direttamente o mediante plug-ins (servlet engines), dalla maggior parte dei server web in circolazione, commerciali e free.
- *Convenienza*. Molti server web che supportano le servlet sono free o comunque hanno un basso costo. Se si dispone già di un server web commerciale che non supporta le servlet l'aggiornamento con appositi plug-ins per il supporto delle servlet è normalmente free.

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(ServletConfig config). Qui avvengono le inizializzazioni delle variabili globali, procedura che viene fatta una sola volta durante l'intero ciclo di vita della Servlet, e viene caricato l'oggetto ServletConfig che potrà poi essere recuperato più tardi mediante il metodo getServletConfig(). Quest'ultima operazione viene eseguita nel metodo init della classe GenericServlet ed è per questo che in ogni classe che la estende, come lo è HttpServlet, all'inizio del metodo init c'è una chiamata al metodo della superclasse e cioè super.init(config). 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.

Nei prossimi paragrafi verranno analizzati i dettagli implementativi del software. A tale scopo si partirà da un livello di descrizione astratto in modo da illustrare le funzionalità del sistema così come vengono percepite dall'utente. Successivamente si passerà alla descrizione dei dettagli implementativi del sistema software in esame.

3.1 Interazione Utente-Sistema

Il sistema in esame è stato sviluppato in modo da permettere ad un utente di effettuare operazioni quali:

- Ottenere una lista delle stazioni attive nella rete sismica dell'OV;
- Salvare su file i segnali relativi a particolari stazioni, selezionate dalla lista di quelle attive;
- Mostrare su display i segnali relativi a particolari stazioni, selezionate dalla lista di quelle attive ed effettuare operazioni di "zoom" e picking;
- Inviare le informazioni di picking al sistema localizzazione automatica, precedentemente descritto, per innescare le procedure di localizzazione.

L'interazione Utente-Sistema viene descritta, a livello macroscopico, tramite diagrammi dei casi d'uso, grazie ai quali, è possibile catturare le funzionalità del sistema così come percepite dall'utente.

In figura 3.1 è illustrato il diagramma dei casi d'uso associato al sistema in esame.

Tale diagramma evidenzia tutte le azioni che un utente può compiere.

In particolare, tramite azioni svolte direttamente su una "console di comando", realizzata tramite applet Java, un utente compie azioni quali:

- creare una lista di stazioni attive, fornita dal Wave Server;
- creare files contenenti le forme d'onda relative a tutte le stazioni, selezionate da console;

- visualizzare le forme d'onda relative a tutte le stazioni, selezionate da console;
- localizzare l'eventuale evento sismico, che le forme d'onda evidenziano.
  L'ultimo punto viene realizzato tramite una retroazione verso la coppia Earthworm-WBSM, in modo da innescare le operazioni di localizzazione in base alle informazioni fornite dall'analisi dell'utente sui segnali.

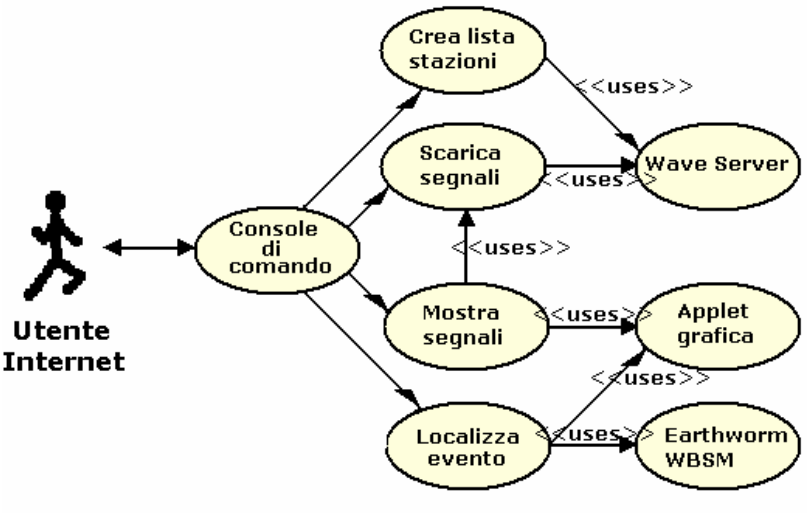


figura 3.1

Allo scopo di realizzare questi quattro punti, sono state sviluppate quattro servlet, ognuna delle quali in grado di svolgere delle particolari operazioni. In questo modo si ottiene un'applicazione complessiva in grado di sfruttare a pieno tutti i pregi di un sistema modulare. Infatti basta sostituire una singola servlet per sostituire la funzionalità che quest'ultima porta al sistema, senza dover cambiare l'intera applicazione. Ciò è giustificato dal fatto che ogni servlet realizzata è del tutto indipendente.

3.1.1 Caso d'uso: Creare lista stazioni

Un utente può ottenere una lista della stazioni attive inserendo da console alcune informazioni riguardanti la modalità di interazione con il Wave Server, modalità vincolate dal protocollo di comunicazione proprio di quest'ultimo. In particolare, tramite una richiesta del tipo: "MENU: *<identificativo richiesta>"* è possibile ottenere una lista di tutte le stazioni attive in rete.

La comunicazione tra "console" e Wave Server avviene mediante connessione socket su protocollo TCP, connessione, cioè, tra una servlet Java ed il Wave Server stesso.

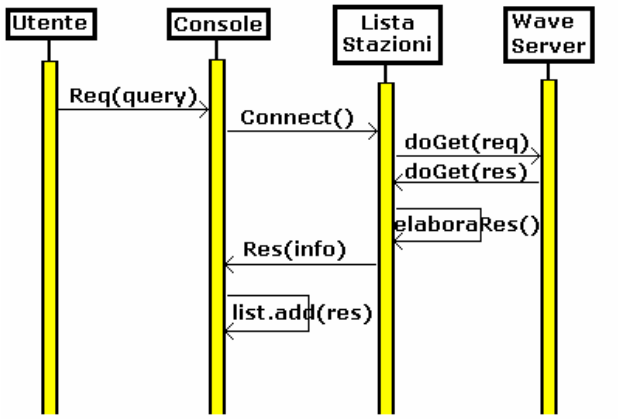


figura 3.2

Il caso d'uso può essere rappresentato dal seguente scenario di interazione (figura 3.2):

- l'utente del sistema richiede una lista di tutte le stazioni al momento disponibili, fornendo la console delle "informazioni di query" su esposte;
- il sistema richiede le informazioni al Wave Server;
- il Wave Server fornisce quanto richiesto;
- il sistema notifica all'utente le informazioni ottenute.

3.1.2 Caso d'uso: Scaricare forme d'onda

Un utente può creare un file contenente le forme d'onda delle stazioni richieste, inserendo da console alcune informazioni riguardanti la modalità di interazione con il Wave Server. Queste modalità sono vincolate dal protocollo di comunicazione del Wave Server stesso. In particolare, inserendo in console sia lo start time" che l' "end time", in modo da selezionare i limiti temporali inferiore e superiore della forma d'onda da scaricare e agendo sul pulsante "Scarica Segnali", viene effettuata, verso il Wave Server, la seguente richiesta: "GETSCN: *<identificativo richiesta> <station> <component> <network> <startTime> <endTime> <fill-value>"*, per poter ottenere i campioni del segnale stesso in formato ASCII.

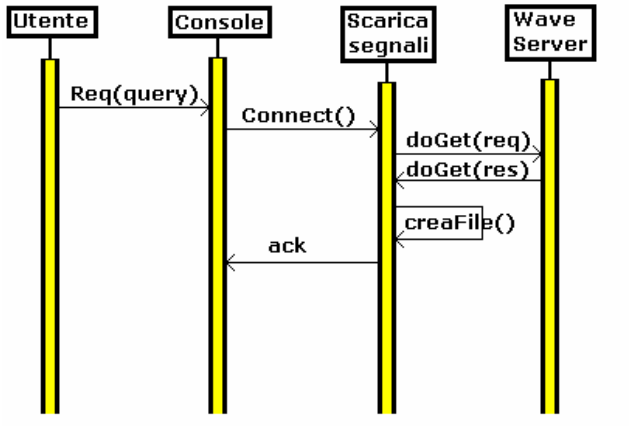


figura 3.3

La comunicazione tra "console" e Wave Server avviene, come già specificato, tramite connessione socket su protocollo TCP, tra una servlet Java ed il Wave Server stesso, così da far sembrare all'utente di colloquiare direttamente con il Wave Server, tramite console.

Il caso d'uso può essere rappresentato dal seguente scenario di interazione (figura 3.3):

- l'utente del sistema richiede la creazione di un file, contenente i campioni delle forme d'onda di tutte le stazioni selezionate sulla console, fornendo la console delle informazioni di query, quali startTime, endTime e fill-value, necessari per una corretta sollecitazione del Wave Server;
- il sistema richiede le informazioni al Wave Server;
- il Wave Server fornisce quanto richiesto;
- il sistema crea il file desiderato.

3.1.3 Caso d'uso: Mostrare segnali a video

Un utente può ottenere la visualizzazione delle forme d'onda relative alle stazioni richieste, inserendo da console le informazioni riguardanti la modalità di interazione con il Wave Server, così come descritto nel precedente paragrafo. Riempendo gli opportuni campi nella console e agendo sul pulsante "Vedi Tracce", viene effettuata, verso il Wave Server, la seguente richiesta: "GETSCN: *<identificativo richiesta> <station> <component> <network> <startTime> <endTime> <fill-value>"*. In tal modo, i campioni delle forma d'onda richiesta saranno mostrati a video, tramite chiamata all'apposito applet di visualizzazione. Come al solito, la comunicazione tra "console" e Wave Server avviene tramite connessione socket su protocollo TCP. In particolare questa connessione è ottenuta tra una servlet Java ed il Wave Server stesso, per ovviare ai vincoli di sicurezza imposti sugli applet, di cui si parlerà in dettaglio nel prossimo capitolo.

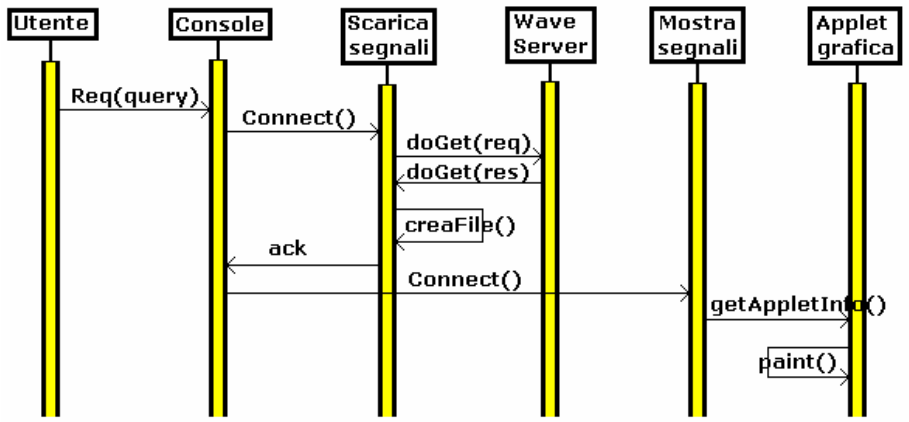


figura 3.4

Il caso d'uso può essere rappresentato dal seguente scenario di interazione (figura 3.4):

- l'utente del sistema richiede dei segnali relativi a specifiche stazioni, fornendo sulla console le informazioni di query, quali startTime, endTime e fill-value, necessari per il corretto funzionamento del Wave Server;
- il sistema richiede le informazioni al Wave Server;
- il Wave Server fornisce quanto richiesto;
- viene creato un file temporaneo, per effettuare analisi off-line sui segnali che si otterranno;
- si notifica alla console l'avvenuta creazione del file;

- si forniscono all'applet di visualizzazione i parametri di inizializzazione, quali il percorso del file creato e il numero di segnali richiesti;
- il sistema mostra sul display i segnali desiderati.

3.1.4 Caso d'uso: Localizzazione di un evento

Un utente può ottenere la localizzazione di un evento, agendo direttamente sull'applet di visualizzazione delle tracce. Infatti, selezionati tutti i parametri necessari allo scopo, quali "istanti di picking", "durata dell'evento", ecc., e inviati tali parametri a Earthworm nel giusto formato tramite connessione socket su UDP, vengono innescate tutte le procedure necessarie; solo se le informazioni dell'utente sono valide, l'evento selezionato viene localizzato e, come già precedentemente illustrato, visualizzato tramite WBSM.
Il caso d'uso può essere rappresentato dal seguente scenario di interazione (figura 3.5):

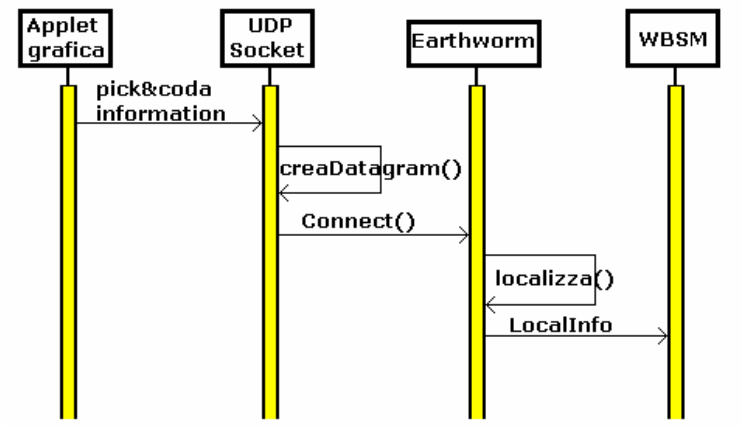


figura 3.5

- l'utente preleva dai sismogrammi le informazioni per la localizzazione;
- vengono preparati i datagrammi con le informazioni utili;
- i datagrammi vengono inviati al modulo "coax2ring" di Earthworm, identificato dall'indirizzo IP della macchina su cui risiede e dal numero di porta su cui ascolta le richieste;
- vengono attivate tutte le procedure di localizzazione tipiche di Eartworm;
- se localizzato, l'evento viene visualizzato tramite "WBSM".

3.2 Diagramma delle classi

Nel presente paragrafo, si vogliono rappresentare gli stessi concetti precedentemente esposti, ma ad un livello di superiore. Grazie ai diagrammi delle classi verranno catturati i concetti del dominio oggetto di studio, indipendentemente dalle scelte implementative: l'attenzione sarà rivolta alla comprensione di "cosa" il sistema software dovrà fare e non di "come" esso lo farà. Verrà, dunque, analizzata la struttura statica del sistema in esame, evidenziando le classi, la loro struttura interna e le loro relazioni. In figura 3.6 è illustrato il diagramma delle classi relativo alle servlet. La relazione di aggregazione è intesa in senso "lasco", in quanto gli oggetti contenuti in "HttpServlet" (il contenitore) possono esistere anche indipendentemente. Ciò comporta la non responsabilità da parte del contenitore per la creazione e distruzione dell'oggetto contenuto. Una volta individuati gli oggetti software che verranno usati per realizzare l'applicazione, le interazioni tra entità principali diventano interazioni tra oggetti e vengono anch'esse rappresentate mediante opportuni diagrammi.

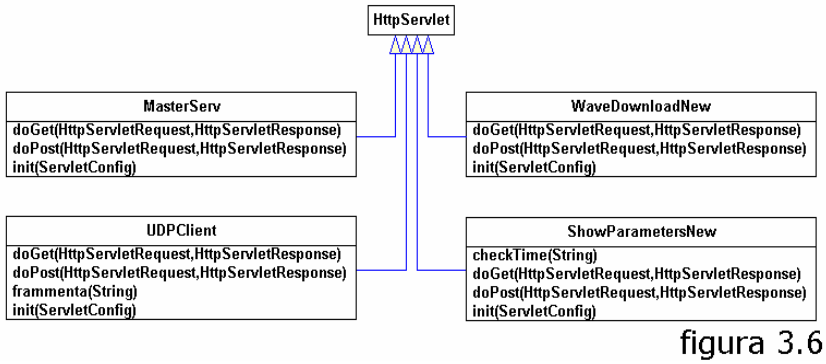


figura 3.6

3.3 ShowParametersNew.java

Nei due diagrammi che seguono sono mostrati i messaggi scambiati nel tempo tra gli oggetti partecipanti alla procedura "init" e "doGet". In particolare in figura 3.7 è rappresentato il diagramma temporale del metodo init, in cui viene letto il file di configurazione "param.properties".

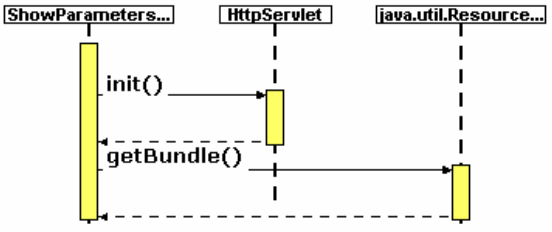


figura 3.7

In figura 3.8 è rappresentato il diagramma temporale del metodo "doGet", che

viene richiamato dal metodo “doPost”.

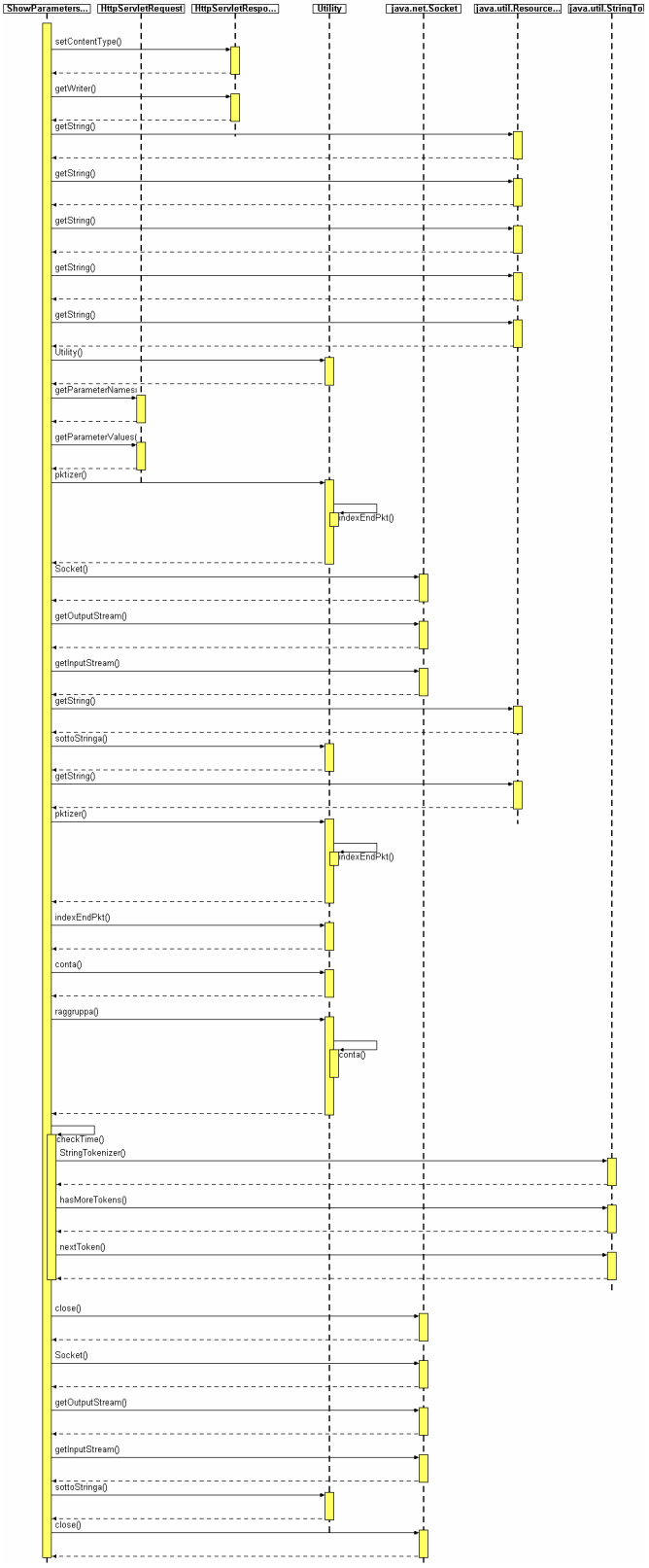


figura 3.8

I metodi *pktizer(), sottoStringa(), indexEndPkt(), conta(), raggruppa(),* fanno tutti parte di un package di utility, creato ad hoc per esigenze di sviluppo, rappresentato in figura 3.9. Grazie a queste utility, è stato possibile selezionare da un'unica stringa delle opportune sottostringhe, quelle, cioè, evidenziate, nel capitolo introduttivo, sulla console, relative alle singole stazioni da selezionare. Il metodo *checkTime()* invece, converte l'orario in millisecondi dal 1970 in una data più intellegibile. Le prime due servlets rappresentano il punto cardine dell'interazione "console-WaveServer", in quanto esse sono il mezzo tramite il quale l'applet "console" può aprire una connessione socket verso il WaveServer ed effettuare le opportune richieste. Esse assumono il ruolo di coordinatore della procedura di comunicazione con il WaveServer e questo spiega l'utilizzo del costruttore "Socket()", appartenente al package "java.net": viene istanziato un oggetto "socket" per aprire una connessione dedicata tra cliente (la servlet) e servente (il WaveServer).

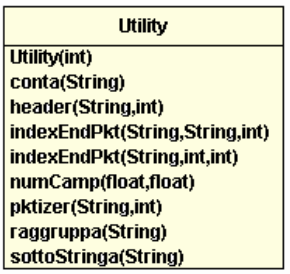
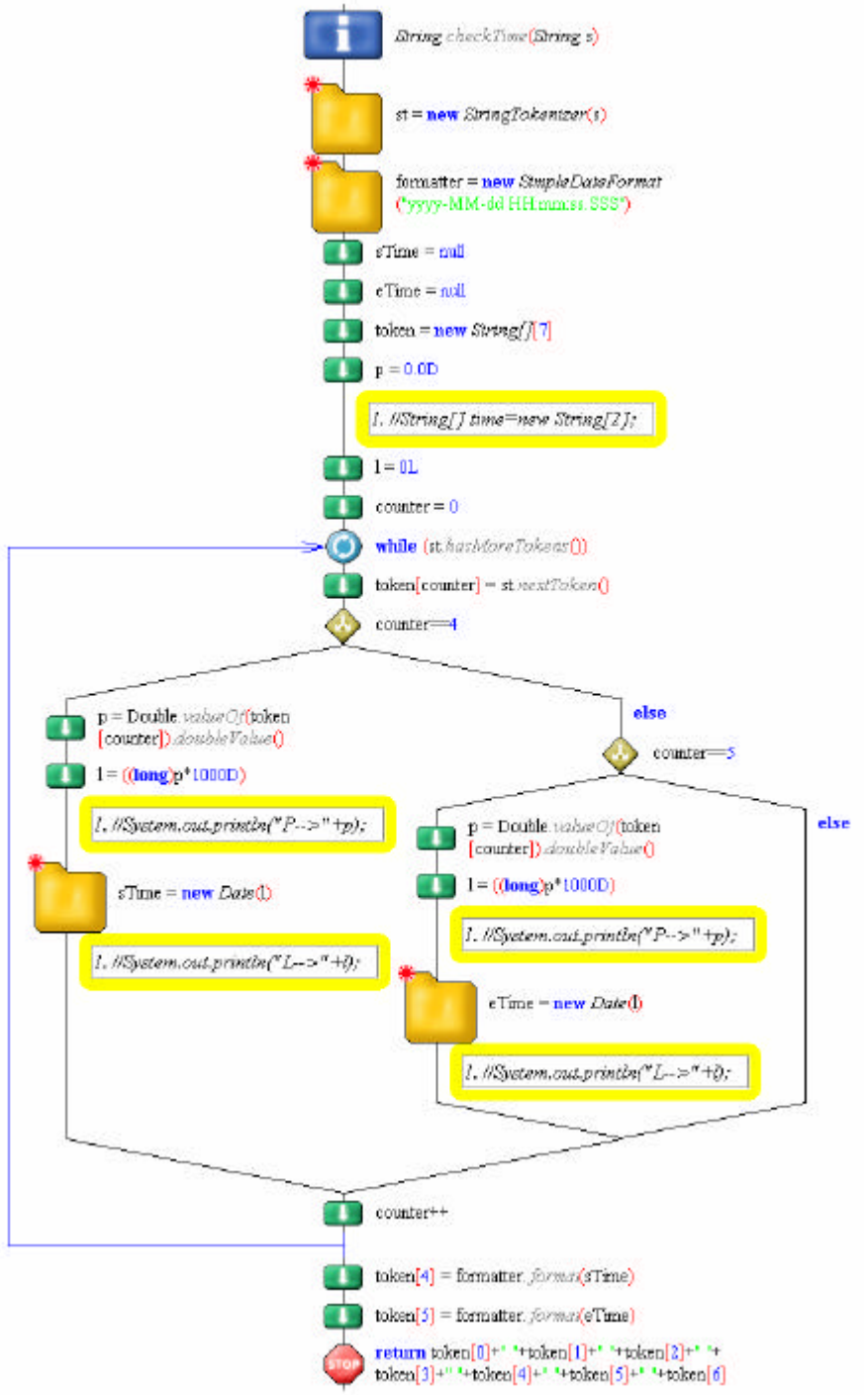


figura 3.9

In particolare, in figura 3.8 si evidenziano i passi fondamentali, che caratterizzano la servlet ShowParametersNew, necessaria per le richieste della lista delle stazioni attive verso il WaveServer:

- configurazione del tipo di contenuto MIME nella risposta;
- lettura delle informazioni di parametrizzazione della servlet da un file del tipo "*configurazione.**properties***", in modo da rendere la servlet stessa configurabile in maniera dinamica, cioè senza dover ricompilare il file sorgente della servlet stessa o riavviare il web-server. Ciò grazie alla classe ResourceBundle;
- lettura della richiesta, tramite "getParameterNames()", proveniente dall'applet "console", vista come cliente di questa servlet;
- apertura di una connessione socket verso uno specifico indirizzo IP e numero di PORTA, informazioni contenute nel file ".properties" di cui sopra;
- apertura di uno stream di uscita verso la macchina di destinazione;
- apertura di uno stream di ingresso, per accettare i messaggi entranti dalla macchina su specificata;
- immagazzinamento delle informazioni ottenute;
- chiusura della connessione.

Come si può notare dalla figura 3.8, tale servlet fa uso del metodo "checkTime()". Esso viene impiegato per trovare i campi riguardanti lo "start-time" e l'"end-time" nella risposta e convertirli da millisecondi dal 1970 in orario GMT. Di seguito sono riportati i dettagli implementativi di tale metodo.



checkTime()

3.4 WaveDownloadNew.java

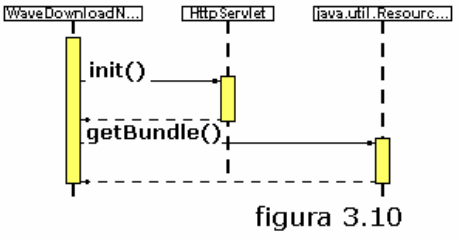


figura 3.10

In figura 3.10, così come illustrato per la precedente servlet, vengono evidenziate le caratteristiche temporali del metodo init di WaveDownloadNew, in cui viene inizializzata la superclasse HttpServlet e letto il file di configurazione "param.properties"

In figura 3.11 vengono, invece, illustrati i passi fondamentali che caratterizzano il metodo "doGet()", necessario per le richieste delle forma d'onda relative alle stazioni selezionate tra quelle presenti nella lista stazioni:

- configurazione del tipo di contenuto MIME nella risposta;
- lettura delle informazioni di parametrizzazione della servlet da un file del tipo "configurazione.***properties***", in modo da rendere la servlet stessa configurabile in maniera dinamica, cioè senza dover ricompilare il file sorgente o riavviare il web-server: ciò grazie alla classe ResourceBundle;
- lettura della richiesta, tramite "getParameterNames()", proveniente dall'applet "console", vista come cliente di questa servlet;
- apertura di una connessione socket verso uno specifico indirizzo IP e numero di PORTA, informazioni contenute nel file ".properties" di cui sopra;
- apertura di uno stream di uscita verso la macchina di destinazione;
- apertura di uno stream di ingresso, per accettare i messaggi provenienti dalla macchina su specificata;
- lettura dello stream in ingresso e creazione del file contenente i campioni delle forme d'onda. Si è scelto di adottare questa tecnica per poter avere a disposizione il segnale per un'analisi off-line, in quanto, come già detto, il contenuto del WaveServer cambia nel tempo ;
- chiusura della connessione.

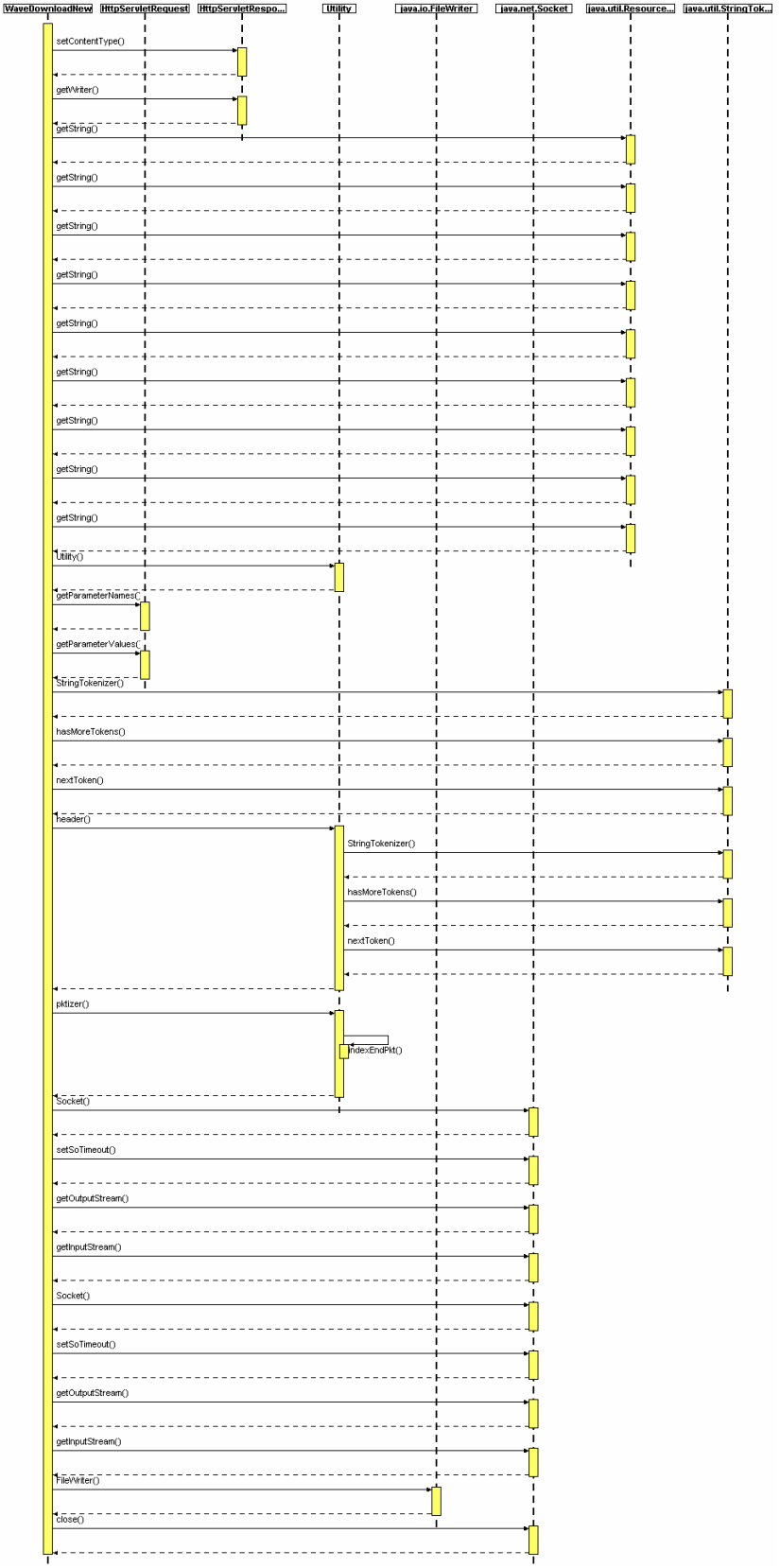

WaveDownloadNew
HttpServletRequest
HttpServletRespo...
Utility
java.io.FileWriter
java.net.Socket
java.util.Resource...
java.util.StringTok...
setContentType()
getWriter()
getString()
getString()
getString()
getString()
getString()
getString()
getString()
getString()
getString()
Utility()
getParameterNames()
getParameterValues()
StringTokenizer()
hasMoreTokens()
nextToken()
header()
StringTokenizer()
hasMoreTokens()
nextToken()
pktizer()
indexEndPkt()
Socket()
setSoTimeout()
getOutputStream()
getInputStream()
Socket()
setSoTimeout()
getOutputStream()
getInputStream()
FileWriter()
close()

figura 3.11

3.5 MasterServ.java

La servlet "MasterServ.java", viene illustrata nelle figure 3.12 e 3.13. Essa gioca il ruolo di connettore tra la "console", che pilota le varie richieste, ecc., e l'applet di rendering delle forme d'onda ottenute. In particolare, in figura 3.12 è illustrato il metodo init.

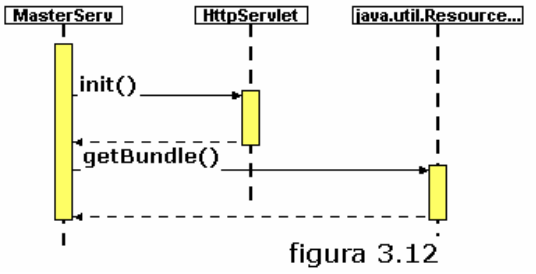


figura 3.12

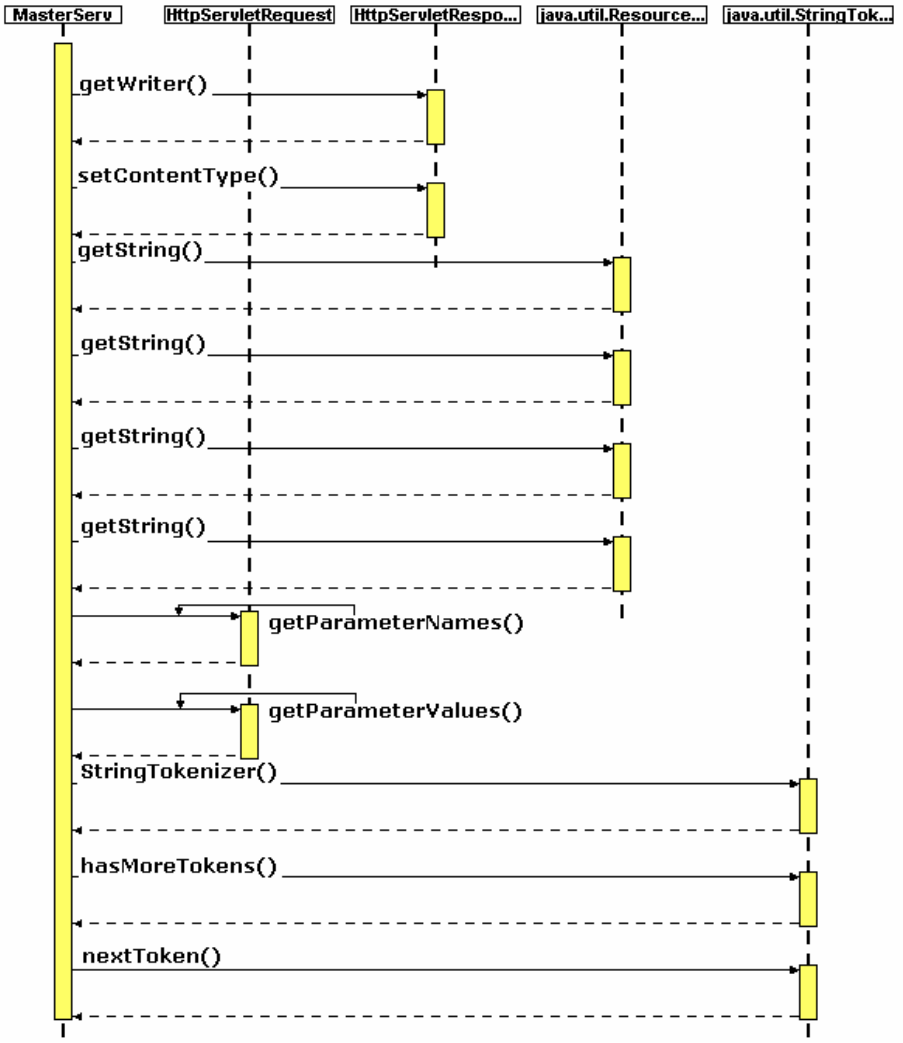


figura 3.13

Essa, come si può notare in figura 3.13, si sviluppa nei seguenti passi:

- configurazione del tipo di contenuto MIME nella risposta;
- lettura delle informazioni di parametrizzazione della servlet da un file del tipo "*configurazione.**properties***", in modo da rendere la servlet stessa configurabile in maniera dinamica, cioè senza dover ricompilare il file sorgente della servlet stessa o riavviare il web-server: ciò grazie alla classe ResourceBundle;
- lettura della richiesta, tramite "getParameterNames()", proveniente dall'applet "console", vista come cliente di questa servlet;
- avvio esecuzione applet grafica.

3.6 UDPClient.java

La servlet "UDPClient.java" viene illustrata nelle figure 3.14 e 3.15. Essa gioca, a differenza della precedente, il ruolo di connettore tra l' "applet dei sismogrammi", su cui l'utente agisce per ottenere la localizzazione di un evento, e i moduli di Earthworm, che effettivamente localizzano l'evento.

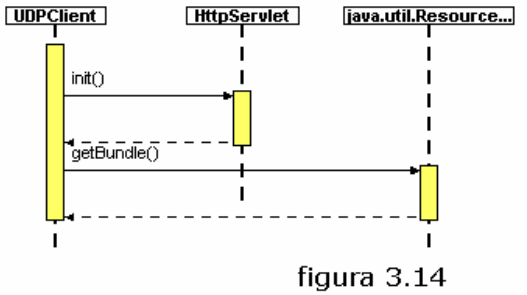


figura 3.14

In figura 3.14 è illustrato il metodo init.

Le operazioni di elaborazione e di connessione vengono implementate nel metodo "doGet" della servlet, che, come illustrato in figura 3.17, si sviluppa nei seguenti passi:

- configurazione del tipo di contenuto MIME nella risposta;
- lettura delle informazioni di parametrizzazione della servlet da un file del tipo "*configurazione.**properties***", in modo da rendere la servlet stessa configurabile in maniera dinamica, cioè senza dover ricompilare il file sorgente o riavviare il web-server: ciò grazie alla classe ResourceBundle;
- lettura dei parametri e dei rispettivi valori nella richiesta, tramite "getParameterNames()" e "getParameterValues()", proveniente dall' "applet dei sismogrammi", vista come cliente di questa servlet;
- elaborazione dei dati ottenuti al passo precedente, tramite il metodo "frammenta()". Tale elaborazione è necessaria in quanto le richieste in ingresso, relative alle singole stazioni, vengono convogliate su un'unica

stringa, ma separate dal carattere speciale "$". In tal caso, come si può notare dalla figura in basso, "frammenta()" separa le richieste in ingresso in base al carattere speciale e le salva in un array di stringhe (NX1), con N pari al numero di richieste da frammentare;

- creazione dei datagrammi "TYPE_PICK2K" e "TYPE_CODA2K" da inviare ad Earthworm. Allo scopo, si costruiscono degli opportuni header, in modo da ottenere l'accesso ai moduli di Earthworm stesso. Infatti, dall'applet di visualizzazione dei sismogrammi vengono creati dei datagrammi contenenti le informazioni di picking, nel giusto formato, che, tramite coax2ring, vengono immessi nel Pick Ring. Questa operazione avvia, poi, la procedura di localizzazione tramite la catena di moduli, precedentemente introdotti. In figura 3.15 e 3.16 sono rappresentati i due tipi di pacchetti che vengono inviati al Pick Ring: il primo trasporta le informazioni sui tempi di arrivo delle fasi, il secondo sulla durata dell'evento.

```
10  4  3 2133 OVO  OVV  U1  20030308183134.90    953    1113    968\n
```

col	length	variable type	description
0	3	unsigned char	Message Type (1-255)
3	3	unsigned char	Module id that produced msg (1-255)
6	3	unsigned char	Installation of origin for this message (1-255)
9	1		unused
10	4	short	Sequence # assigned by picker (0-9999), allows you to match a pick with its coda info
14	1		unused
15	5	char[]	Site code (left justified)
20	2	char[]	Network code (left justified)
22	3	char[]	Component code (left justified)
25	1		unused
26	1	char	Polarity of first break
27	1	char	Quality of pick assigned by picker (0-4) with 0 begin best
28	2		unused (space for phase)
30	17	char[]	Arrival time in the form of: yyyymmddhhmmss.ff
47	8		Amplitude of 1st peak after arrival time
55	8		" 2nd " " " "
63	8		" 3rd " " " "
71	1	char	newline character

figura 3.15: TYPE_PICK2K

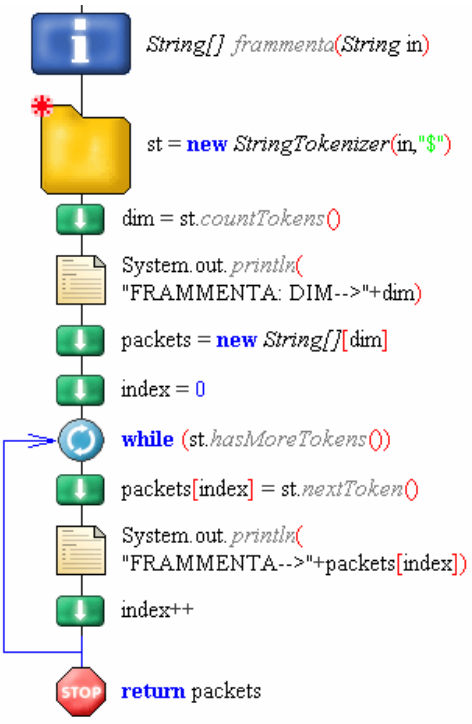
```
11  4  3 2133 OVO  OVV     27    201    276    289      0      0  7\n
```

col	length	variable type	description
0	3	unsigned char	Message Type (1-255)
3	3	unsigned char	Module id that produced msg (1-255)
6	3	unsigned char	Installation of origin for this message (1-255)
9	1		unused
10	4	short	Sequence # assigned by picker(0-9999), allows you to match a pick with its coda info
14	1		unused
15	5	char[]	Site code (left justified)
20	2	char[]	Network code (left justified)
22	3	char[]	Component code (left justified)
25	8	long	Coda window 1 average absolute amplitude
33	8	long	" " 2 " " "
41	8	long	" " 3 " " "
49	8	long	" " 4 " " "
57	8	long	" " 5 " " "
65	8	long	" " 6 " " "
73	4	short	Coda Duration (seconds)
77	1	short	Coda weigth (quality)
78	1	char	newline character

figura 3.16: TYPE_CODA2K

La dimensione è fissa per entrambe i pacchetti, che vengono letti dal modulo eqproc, e pari a 72 e 79 byte, rispettivamente, a cui va aggiunto un header di dimensione, anch'essa fissa, di 6 byte, per il coax2ring.

- invio dei datagrammi, tramite connessione socket su UDP, ad un modulo di Earthworm in ascolto su una determinata porta, con un ritardo di circa un secondo tra una spedizione e l'altra, per agevolare Earthworm nelle elaborazioni delle informazioni in ingresso;
- chiusura della connessione.



frammenta()

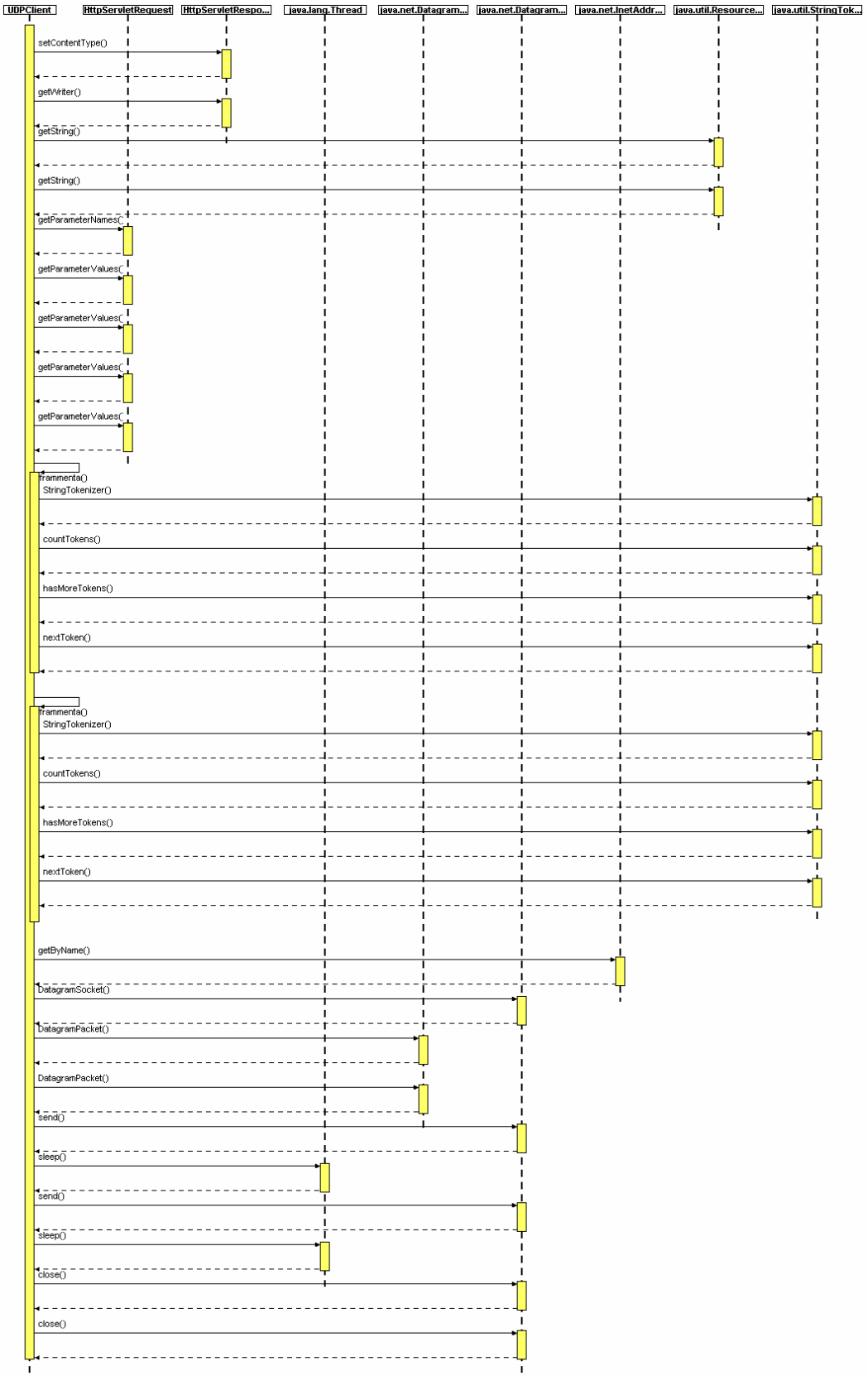

UDPClient
HttpServletRequest
HttpServletRespo...
java.lang.Thread
java.net.Datagram...
java.net.Datagram...
java.net.InetAddr...
java.util.Resource...
java.util.StringTok...
setContentType()
getWriter()
getString()
getString()
getParameterNames()
getParameterValues()
getParameterValues()
getParameterValues()
getParameterValues()
frammenta()
StringTokenizer()
countTokens()
hasMoreTokens()
nextToken()
frammenta()
StringTokenizer()
countTokens()
hasMoreTokens()
nextToken()
getByName()
DatagramSocket()
DatagramPacket()
DatagramPacket()
send()
sleep()
send()
sleep()
close()
close()

figura 3.17

3.3 Dettagli implementativi

Nel presente paragrafo sono illustrate le modalità di processamento dei dati, evidenziando i processi che le variabili di input e output subiscono.

3.3.1 ShowParametersNew.java

In figura 3.18 viene illustrato il diagramma di flusso del metodo di inizializzazione della servlet.

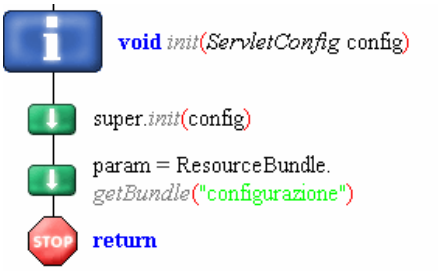


figura 3.18

Come precedentemente ricordato, viene innanzitutto chiamato il metodo init della superclasse al quale viene passato l'oggetto ServletConfig che contiene i parametri della servlet e il riferimento a ServletContext, che rappresenta il contesto su cui gira la servlet. Successivamente, viene letto il file di configurazione "*configurazione.properties".*

In figura 3.19 è illustrato il diagramma di flusso della procedura "doGet", che caratterizza la servlet "ShowParametersNew". Tale figura ha l'unico scopo di evidenziare i rami decisionali che caratterizzano tale procedura. nonché i blocchi elaborativi fondamentali. I particolari saranno illustrati nelle successive immagini.

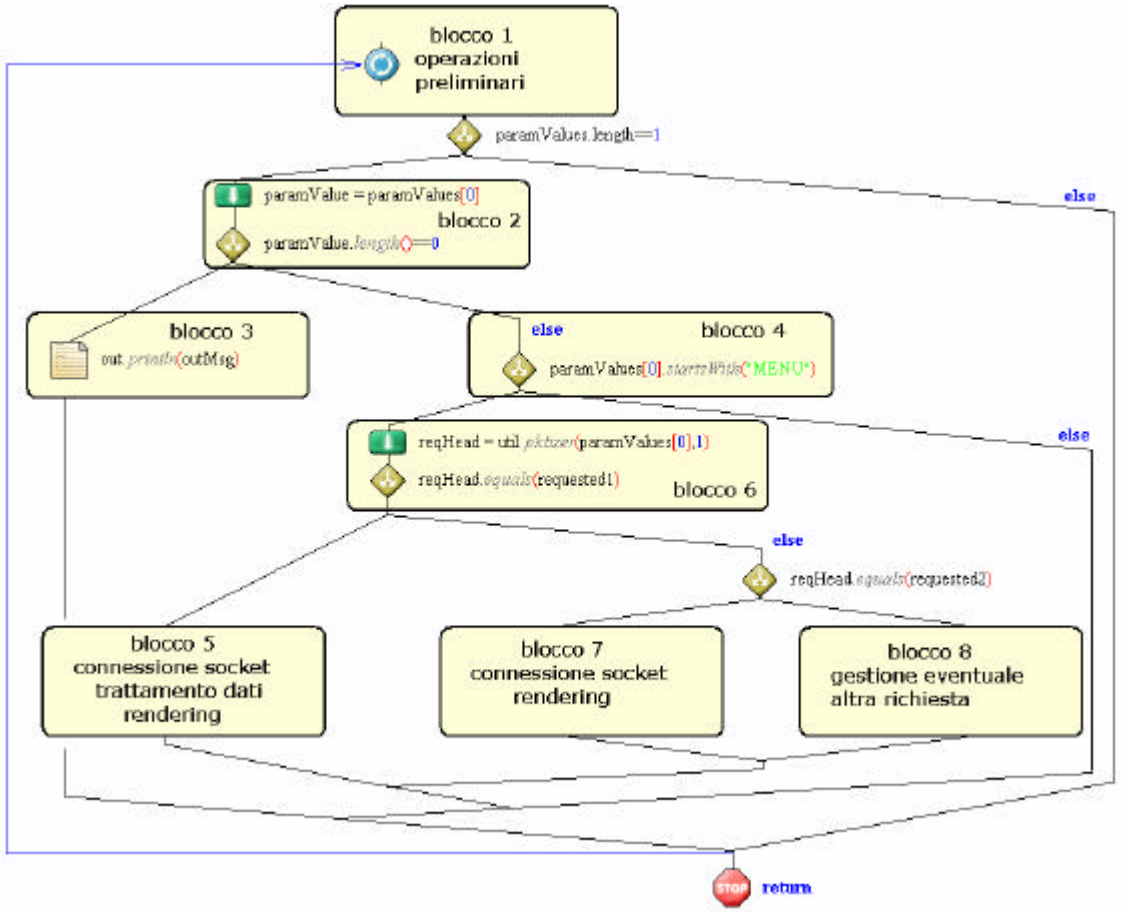


figura 3.19

In figura 3.20 è illustrato in dettaglio il blocco 1, in cui vengono effettuate delle operazioni preliminari per la procedura, come istanziare le variabili da utilizzare e la lettura dei parametri nella query, grazie al metodo "*getParameterNames()*" dell'interfaccia "*javax.servlet.ServletRequest*", che restituisce un *enumerazione* di stringhe rappresentative dei parametri. Il controllo "if" viene effettuato per assicurarsi che ci sia almeno una richiesta.

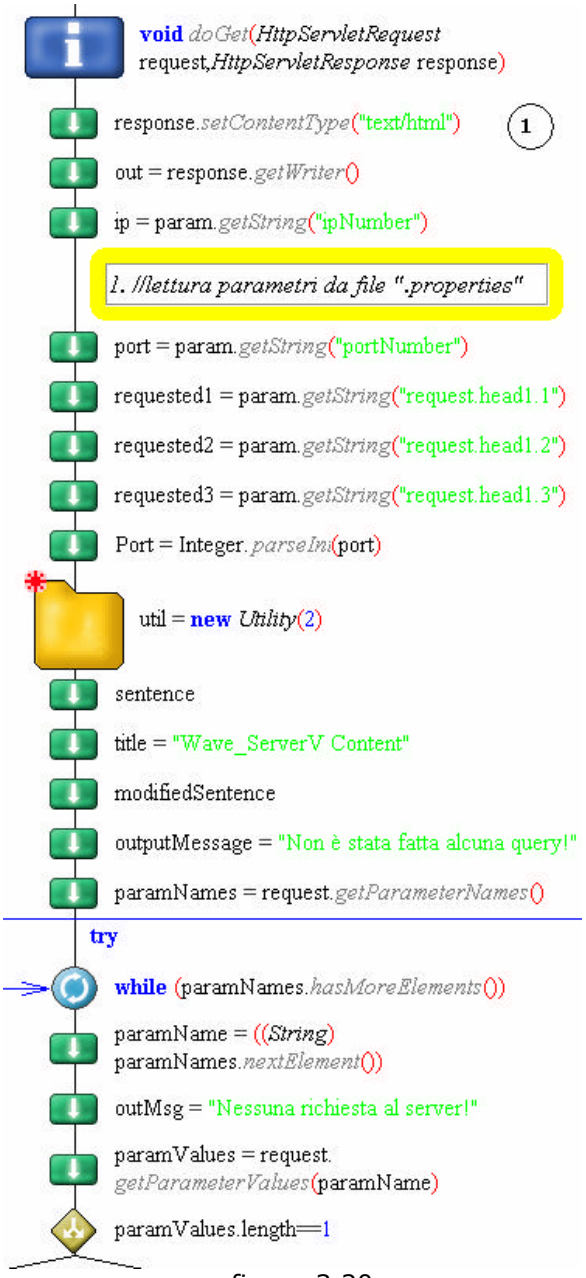


figura 3.20

Vengono inoltre letti i parametri di configurazione quali indirizzo IP della macchina su cui risiede il WaveServer, il numero di porta su cui quest'ultimo attende le richieste ed i tipi di richiesta da accettare.

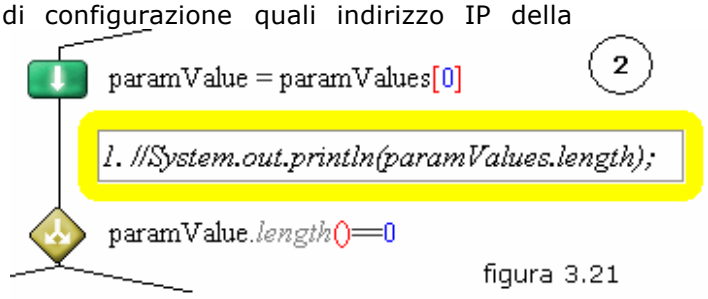


figura 3.21

Nel blocco 2 di figura 3.21 viene effettuato un ulteriore controllo sull'esistenza dell'unico parametro che effettivamente sarà presente in una richiesta.
Il blocco 3 di figura 3.22 viene utilizzato per mandare messaggi di controllo verso la console.

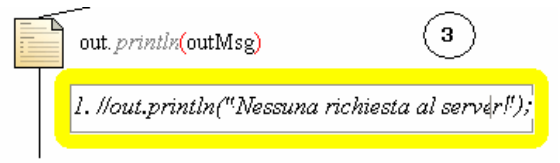


figura 3.22

Il blocco 4 di figura 3.23 rappresenta il caso in cui c'è stata richiesta e questa coincide con quella del tipo "*MENU...*".

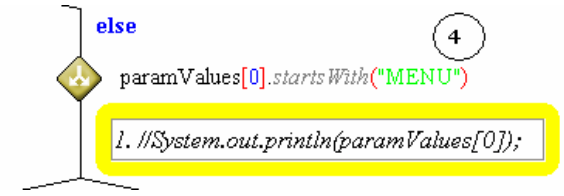


figura 3.23

Una volta effettuati tutti i controlli, riconosciuto il tipo di query e validato il parametro di configurazione letto, così come illustrato nel blocco 6 di figura 3.24, il flusso di dati si dirama al blocco 5, 7 o 8.

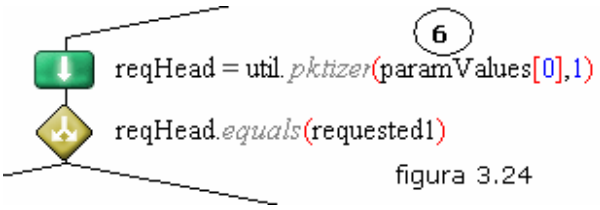


figura 3.24

Nel blocco 5 di figura 3.26, vengono aperti: una connessione via socket verso la destinazione, tramite il costruttore "*Socket(String ip, int port)*" della classe "*java.net.Socket*"; uno stream di input per la lettura delle informazioni da parte dell'utente; uno stream di output verso la destinazione per l'invio delle informazioni elaborate dalla servlet.
Le informazioni richieste vengono fornite dal WaveServer in una stringa su un'unica linea, in formato ASCII. Per questo tali informazioni sono per selezionare i campi relativi ad una singola stazione e salvarli in una riga di un vettore NX1, con N pari al numero delle stazioni di cui il WaveServer tiene traccia. Dopo il trattamento dei dati le informazioni vengono inviate direttamente alla console per il rendering delle informazioni ottenute. Nel blocco

7 di figura 3.25 manca il trattamento dei dati perché il tipo di richiesta in esso gestita produce un'unica stringa relativa ad un'unica stazione, quella richiesta.
Il blocco 8 in figura 3.19 gestisce un tipo di query per la quale il WaveServer è predisposto, ma che attualmente non è implementata.

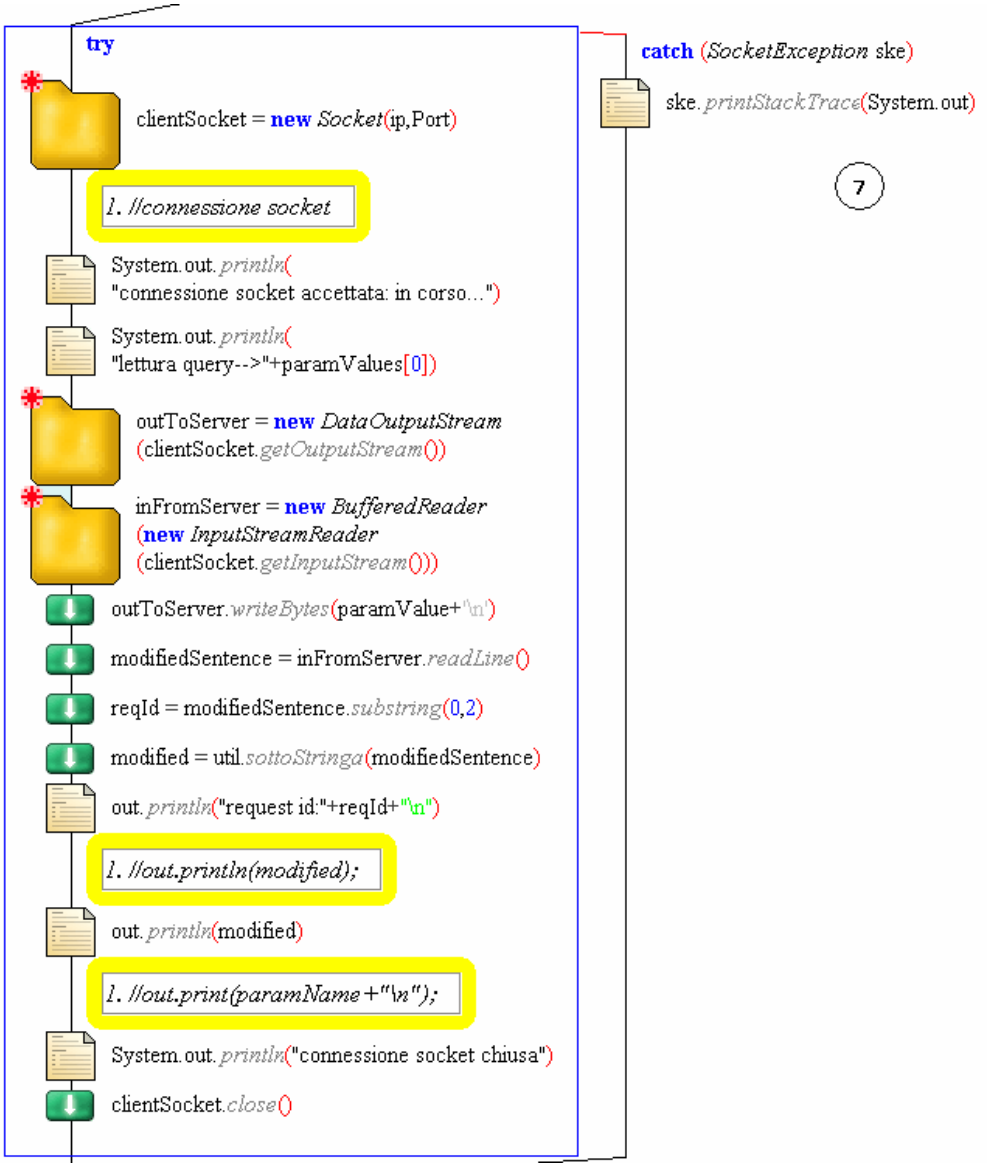


figura 3.25

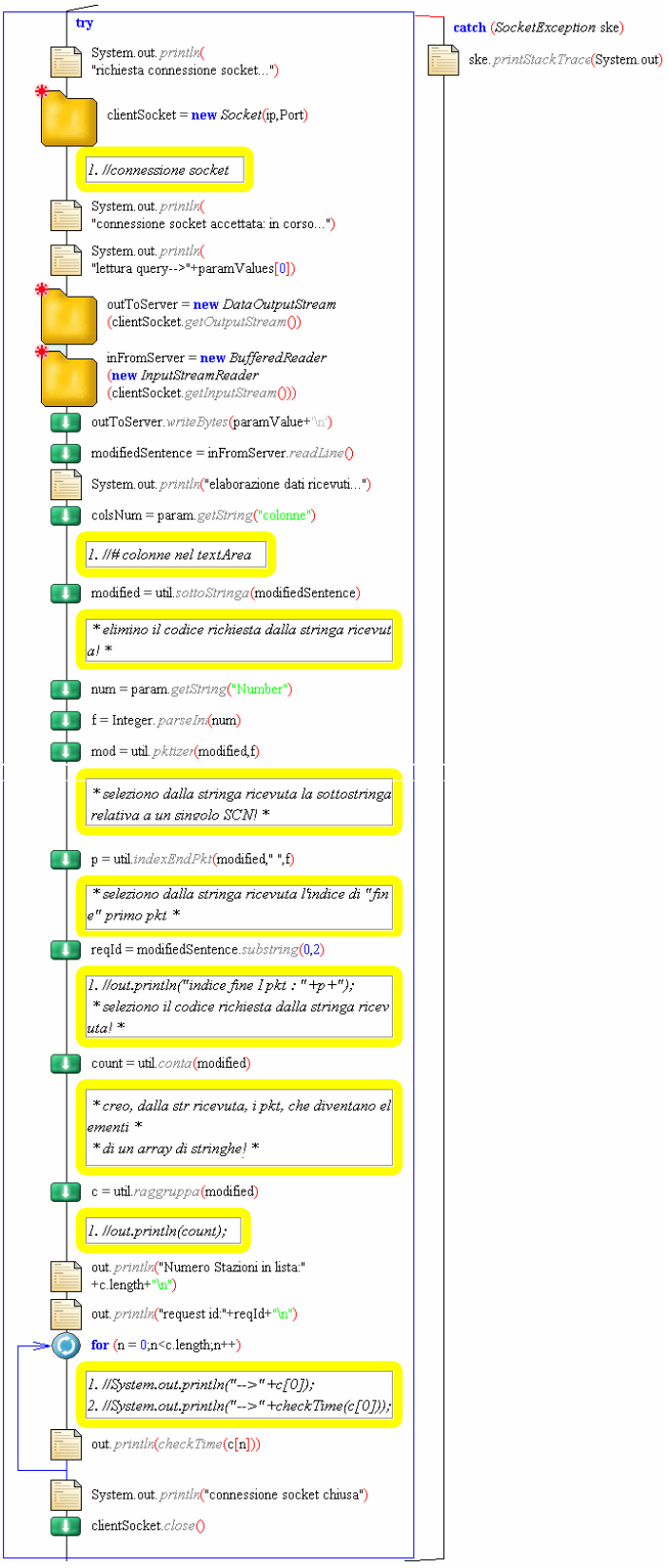


figura 3.26 blocco 5

3.3.2 WaveDownloadNew.java

In figura 3.27 viene illustrato il diagramma di flusso del metodo di inizializzazione della servlet.

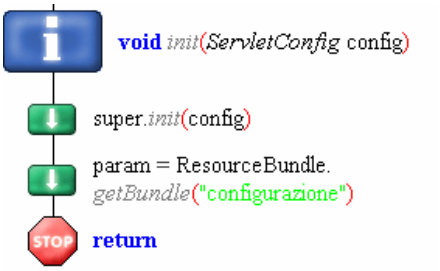


figura 3.27

Come precedentemente ricordato, viene innanzitutto chiamato il metodo init della superclasse al quale viene passato l'oggetto ServletConfig che contiene i parametri della servlet e il riferimento a ServletContext, che rappresenta il contesto in cui gira la servlet. Successivamente, viene letto il file di configurazione "*configurazione.properties".*

In figura 3.28 è illustrato il diagramma di flusso della procedura "doGet", che caratterizza la servlet "WaveDownloadNew". Essa evidenzia i rami decisionali, caratterizzanti tale procedura e sette blocchi fondamentali, che saranno illustrati nelle prossime immagini.

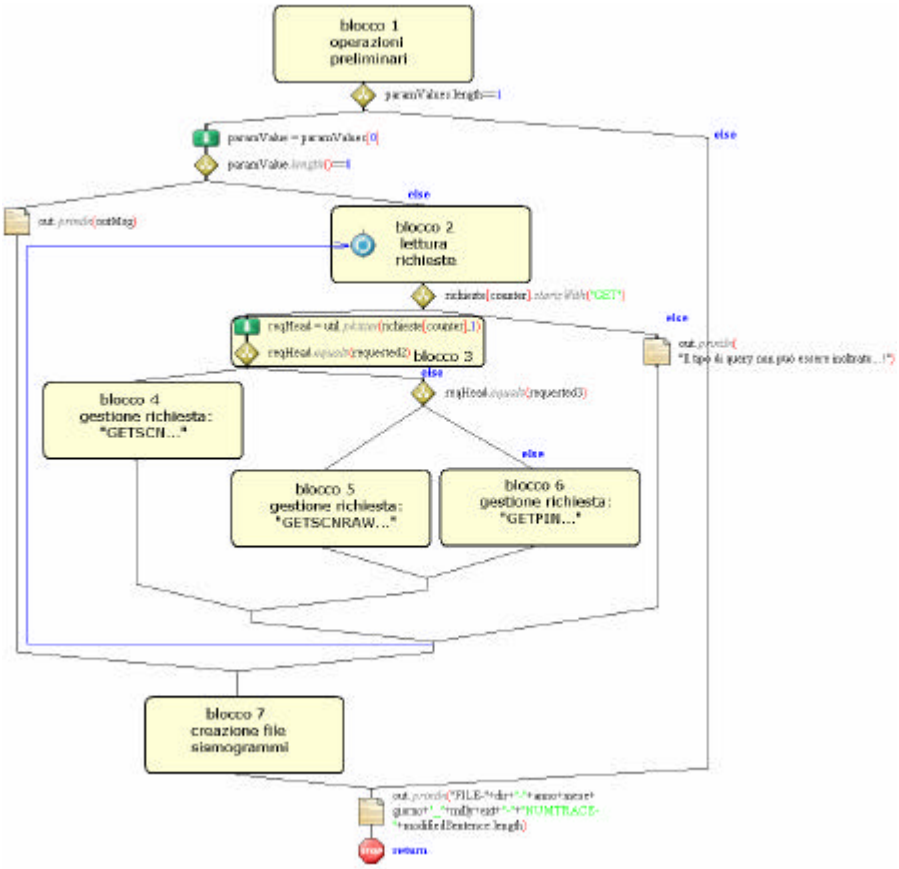


figura 3.28

In figura 3.29 è illustrato in dettaglio il blocco 1.

figura 3.29

In questo blocco vengono effettuate delle operazioni preliminari, come l'inizializzazione delle variabili, la lettura dei parametri di configurazione dal file ".properties" (come i tipi di richiesta da accettare) e la lettura dei parametri nella query, così come fatto nel blocco 1 della precedente procedura. Nel ciclo "while" vengono lette e salvate le richieste sugli elementi di un vettore (NX1). Il controllo "if" viene effettuato per validare la richiesta che tale servlet dovrà eseguire.

In figura 3.30 sono evidenziati i particolari del blocco 2, in cui vengono lette le richieste dal vettore precedentemente inizializzato e riconosciuta il tipo di query da inviare, tra quelle contemplate dal WaveServer.

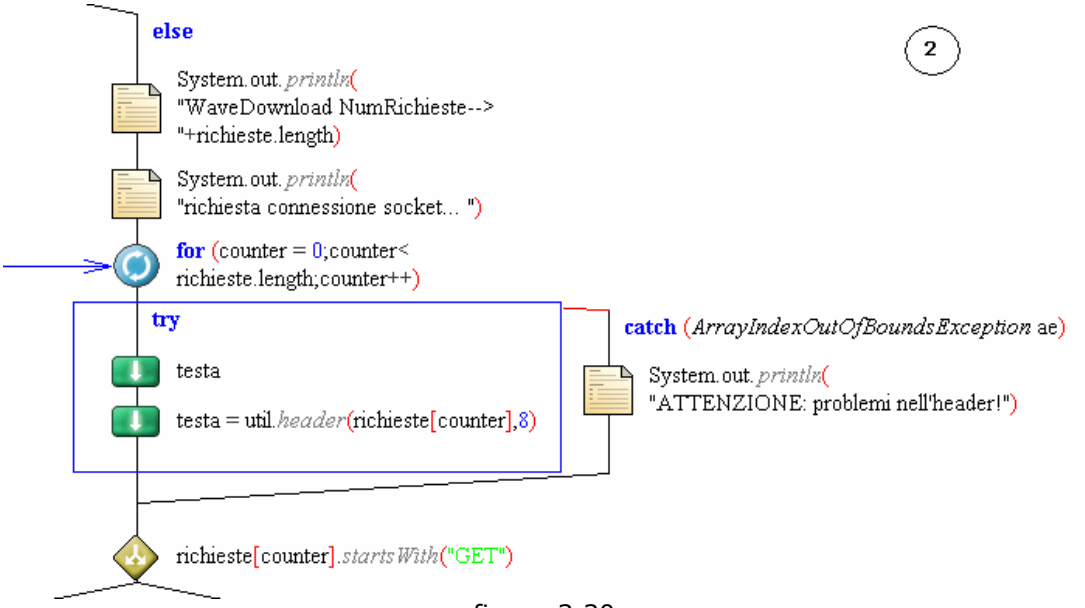


figura 3.30

Come si può notare, qui viene utilizzato un ciclo "for" che conterrà i successivi blocchi 3,4,5,6, per ripetere tutte le operazioni che questi offrono, fintantoché è presente una richiesta nel vettore delle "richieste".

In figura 3.31 viene effettuato un controllo sul tipo di richiesta ricevuta, per poi agire di conseguenza.

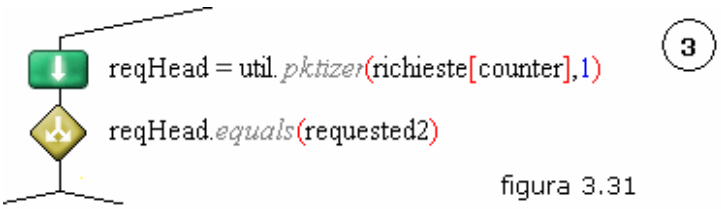


figura 3.31

Infatti, il blocco 4, in figura 3.32, gestisce l'azione conseguente al tipo di richiesta:"GETSCN...".

Questo blocco potrebbe rappresentare l'i-esimo passo del ciclo "for", prima sottolineato.

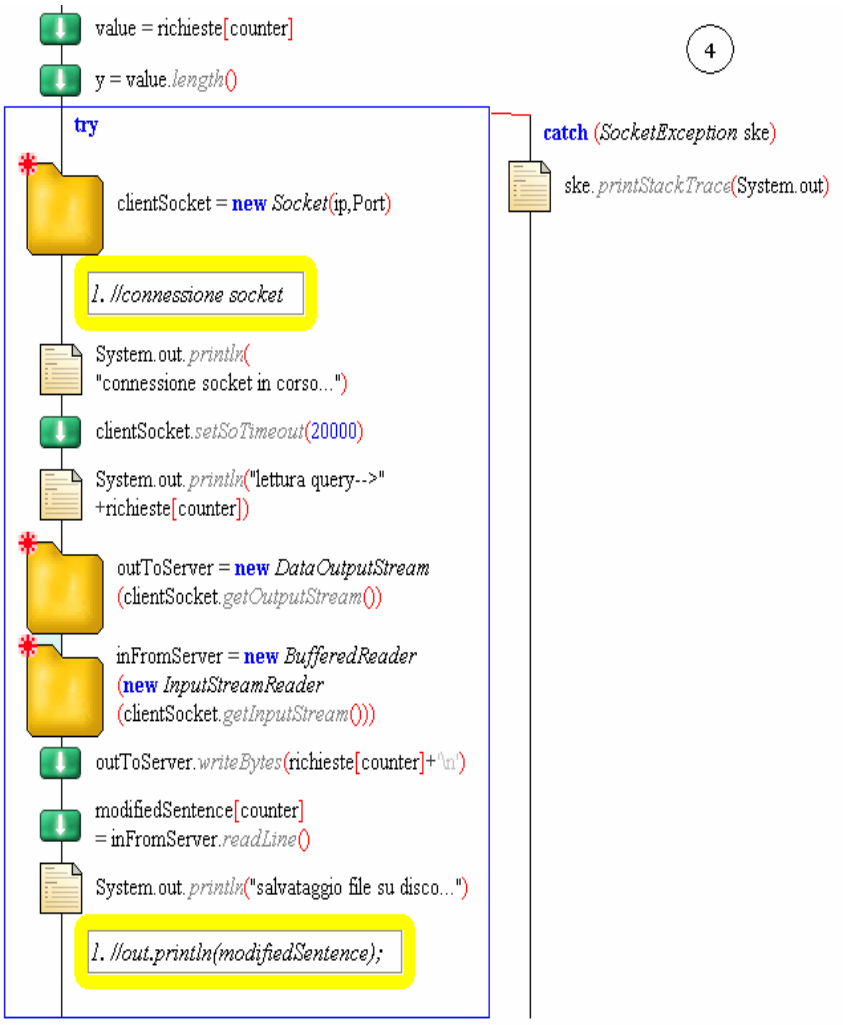


figura 3.32

In questa fase, sono effettuate le seguenti operazioni:

- lettura della richiesta;
- apertura della connessione socket verso la destinazione;
- configurazione del tempo di timeout sulla connessione a venti secondi;
- apertura di uno stream di output *outToServer* ed uno di input i*nFromServer*;
- invio della richiesta su outToServer;
- salvataggio della risposta in inFromServer.

Da notare che la connessione non viene chiusa una volta ottenuta la risposta i-esima, ma viene chiusa solo alla fine del ciclo, quando non ci saranno più richieste da inviare. Nel blocco 5 di figura 3.33, vengono effettuate le stesse operazioni appena descritte, ma con lo scopo di gestire il secondo tipo di query,

mentre il blocco 6 è presente per gestire il terzo tipo di query che attualmente viene ignorato dal WaveServer.

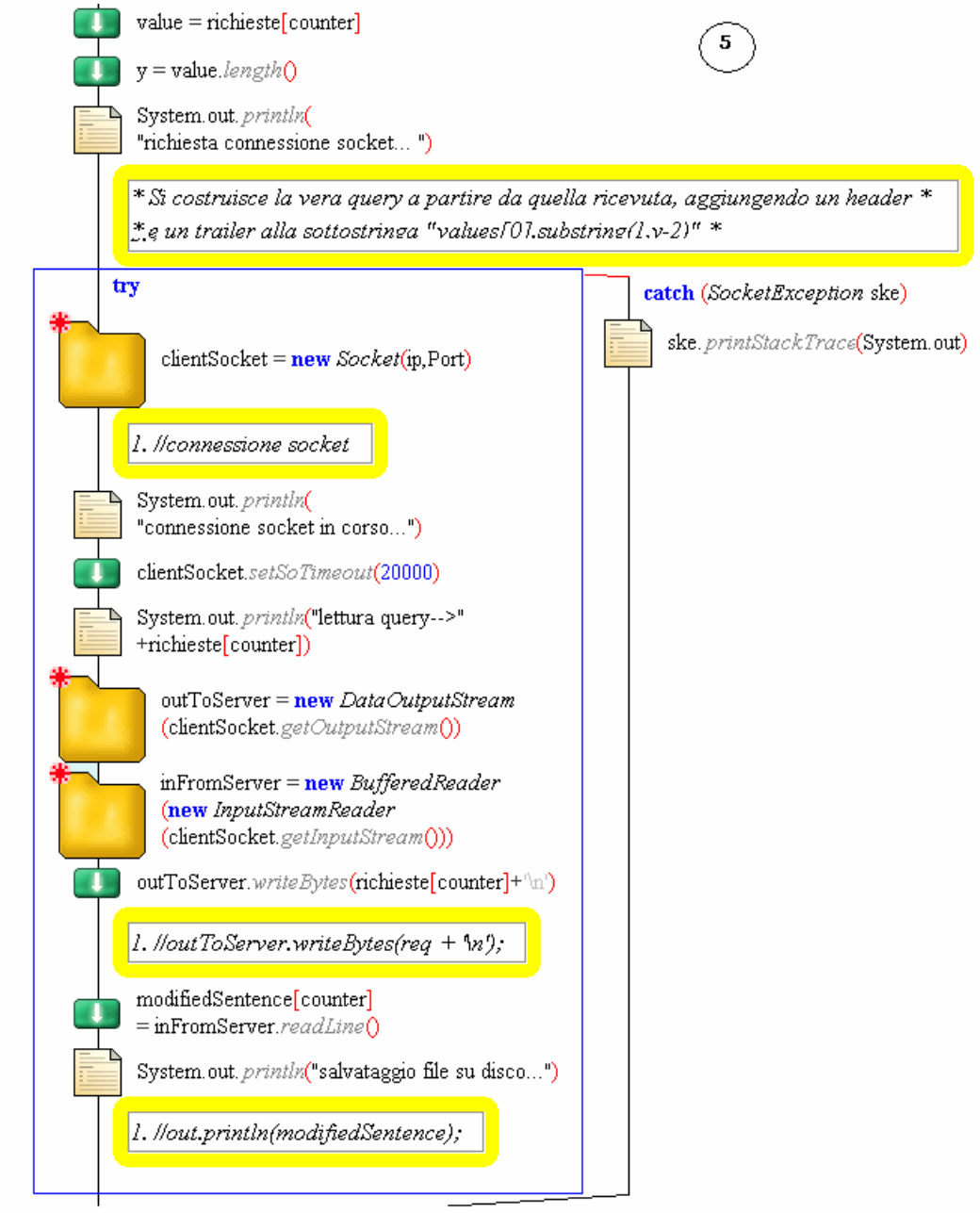


figura 3.33

Infine, in figura 3.32, viene illustrato il blocco 7, necessario per la creazione del file che conterrà le forme d'onda richieste. Un tipico file è mostrato in figura, dove il simbolo "SG" è l'identificativo della richiesta ed usato per distinguere le informazioni relative ad una stazione da quelle di un'altra: una sorta di "multiplex delle risposte" provenienti dal WaveServer.

```
SG 0 ASO V OV F i2 1047972599.990000 100.000000 ---campioni--- SG 0
BAC V OV F i2 1047972599.990000 100.000000---campioni--- SG 0 BKE V
OV F i2 1047972599.990000 100.000000---campioni--- SG 0 CPU V OV F
i2 1047972599.990000 100.000000---campioni--- SG 0 DMP V OV F i2
1047972599.990000 100.000000---campioni---
```

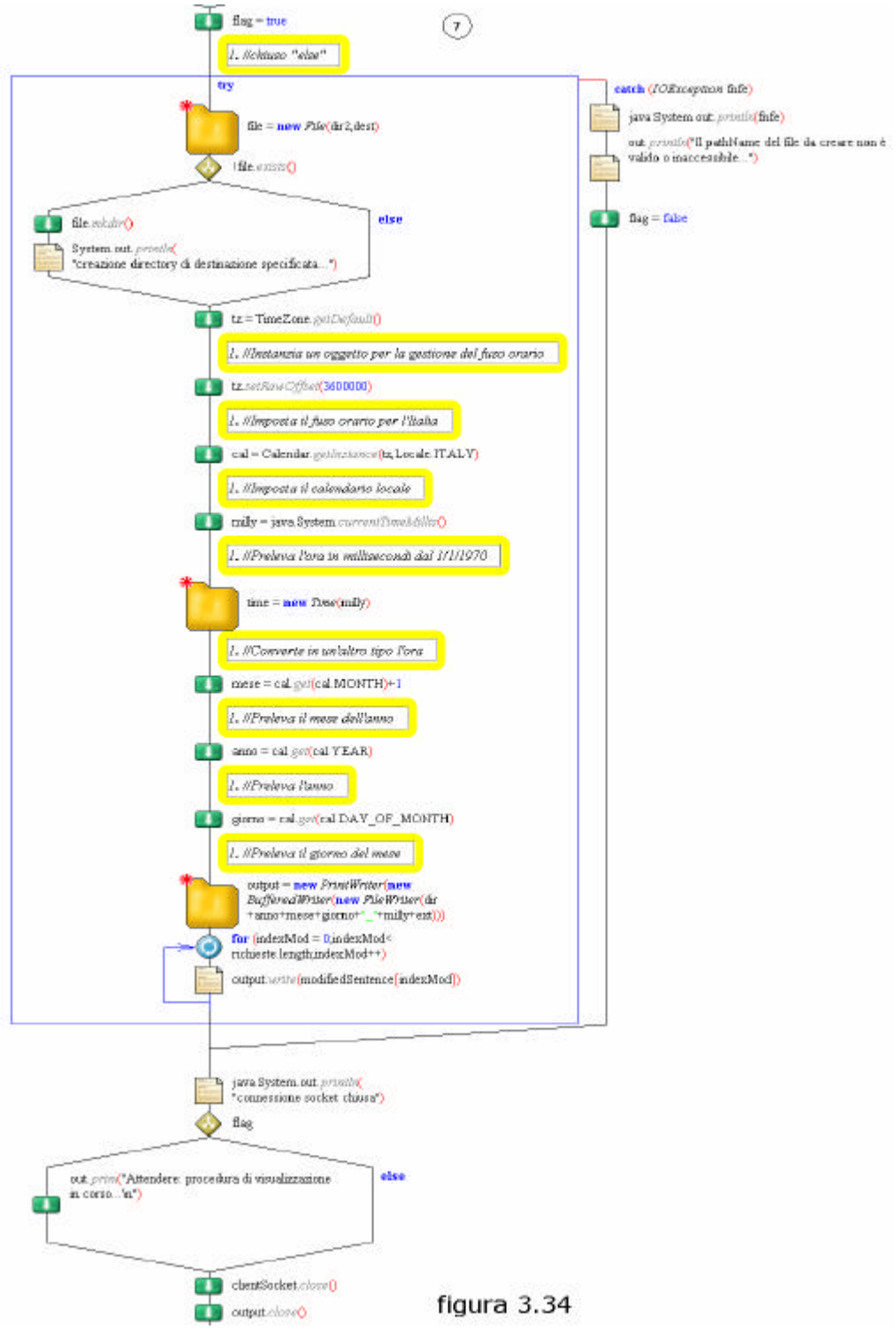


figura 3.34

Il nome al file creato viene dato in base all'ora di creazione, tradotta in millisecondi dal 1970, in modo da distinguere files creati in istanti di tempo diversi e permettere il loro reperimento per un'analisi off-line.
Inoltre, per gestire la creazione del file stesso viene inizializzata a "true" una variabile booleana "flag". Nel caso in cui venga intercettata un eccezione del tipo "IOException", questa variabile viene posta a "false" per indicare la non riuscita della creazione del file stesso. Solo dopo queste operazioni, viene chiusa la connessione socket e salvato il file.

3.3.3 MasterServ.java

In figura 3.35 è illustrato il diagramma di flusso della procedura "doGet", che caratterizza la servlet "MasterServ". Essa, così come precedentemente ricordato, evidenzia i rami decisionali, caratterizzanti tale procedura e due blocchi fondamentali, che saranno illustrati, nei particolari, nelle prossime immagini.

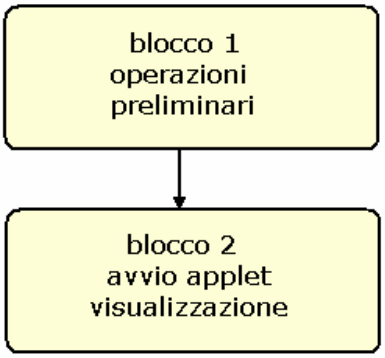


figura 3.35

In figura 3.36 viene illustrato il diagramma di flusso del metodo di inizializzazione della servlet.
Come precedentemente ricordato, viene innanzitutto chiamato il metodo init della superclasse al quale viene passato l'oggetto ServletConfig che contiene i parametri della servlet e il riferimento a ServletContext, che rappresenta il contesto in cui gira la servlet. Successivamente, viene letto il file di configurazione *"configurazione.properties".*

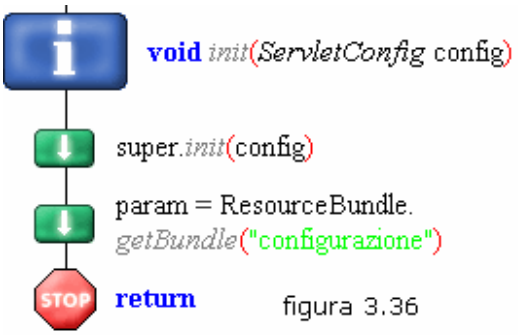


figura 3.36

Tale diagramma descrive, quindi, il funzionamento della servlet in oggetto, il cui scopo è quello di richiamare l'applet di visualizzazione delle tracce.

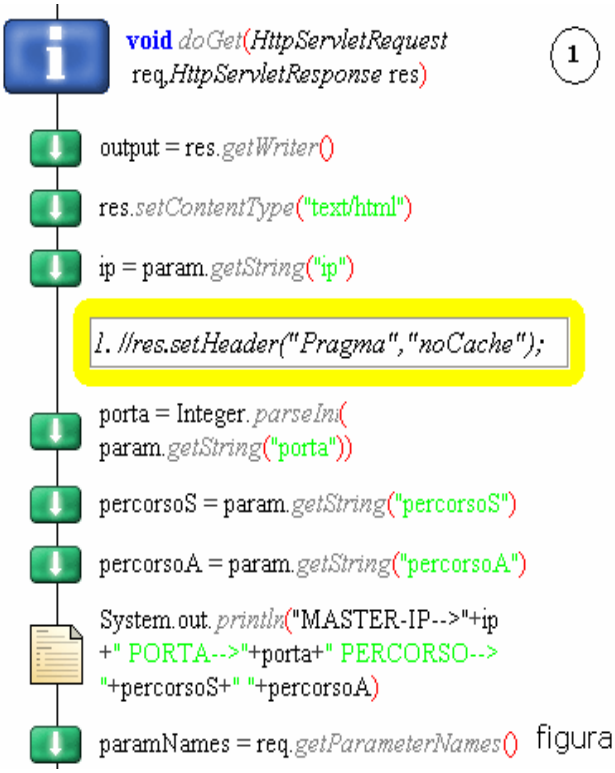


figura 3.37

In particolare, nel blocco 1, in figura 3.37 vengono effettuate delle operazioni preliminari, come l'inizializzazione delle variabili, la lettura dei parametri di configurazione dal file ".properties", come l'indirizzo IP della macchina che fornirà le forme d'onda lette (dove è installato il web server), la porta su cui il web server si aspetta le richieste, il percorso che porta all'esecuzione della servlet in oggetto e quello per richiamare l'applet di visualizzazione e, infine, la lettura delle richieste tramite il metodo "getParameterNames", come già noto.

Nel blocco 2, illustrato in figura 3.38, in base alla lettura dei parametri nella richiesta, viene creato l'URL che punterà al file, precedentemente creato, e generato il codice HTML, necessario per avviare l'applet di visualizzazione.

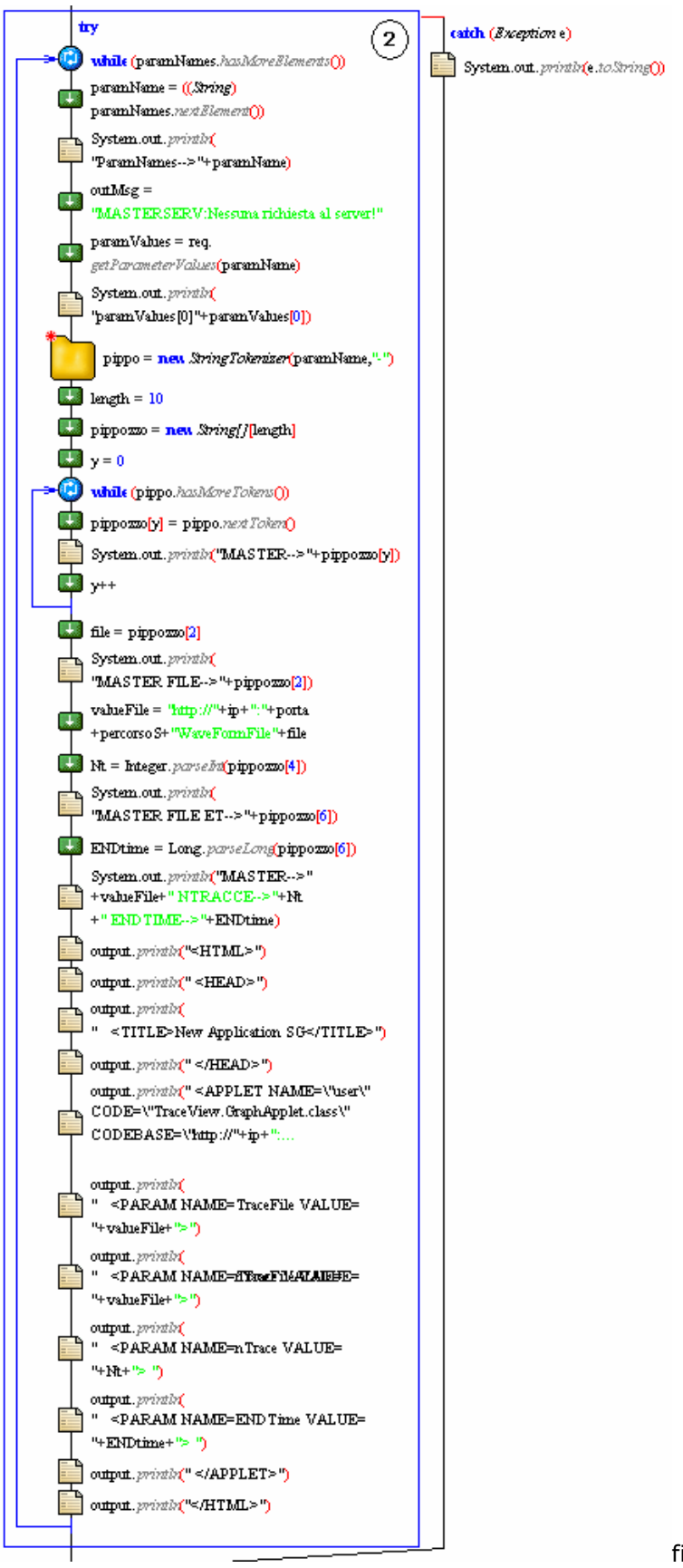

try
2
while (paramNames.hasMoreElements())
paramName = ((String)
paramNames.nextElement())
System.out.println(
"ParamNames-->"+paramName)
outMsg =
"MASTERSERV:Nessuna richiesta al server!"
paramValues = req.
getParameterValues(paramName)
System.out.println(
"paramValues[0]"+paramValues[0])
pippo = new StringTokenizer(paramName,"-")
length = 10
pippozzo = new String[j][length]
y = 0
while (pippo.hasMoreTokens())
pippozzo[y] = pippo.nextToken()
System.out.println("MASTER-->"+pippozzo[y])
y++
file = pippozzo[2]
System.out.println(
"MASTER FILE-->"+pippozzo[2])
valueFile = "http://"+ip+":"+porta
+percorsoS+"WaveFormFile"+file
Nt = Integer.parseInt(pippozzo[4])
System.out.println(
"MASTER FILE ET-->"+pippozzo[6])
ENDtime = Long.parseLong(pippozzo[6])
System.out.println("MASTER-->"
+valueFile+" NTRACCE-->"+Nt
+" END TIME-->"+ENDtime)
output.println("<HTML>")
output.println(" <HEAD>")
output.println(
" <TITLE>New Application SG</TITLE>")
output.println(" </HEAD>")
output.println(" <APPLET NAME=\"user\"
CODE=\"TraceView.GraphApplet.class\"
CODEBASE=\"http://"+ip+"....
output.println(
" <PARAM NAME=TraceFile VALUE=
"+valueFile+"> ")
output.println(
" <PARAM NAME=
"+valueFile+"> ")
output.println(
" <PARAM NAME=nTrace VALUE=
"+Nt+"> ")
output.println(
" <PARAM NAME=END Time VALUE=
"+ENDtime+"> ")
output.println(" </APPLET>")
output.println("</HTML>")
catch (Exception e)
System.out.println(e.toString())

figura 3.38

3.3.4 UDPClient.java

In figura 3.39 è illustrato il diagramma di flusso della procedura "doGet", che caratterizza la servlet "UDPClient". Essa, così come precedentemente ricordato, evidenzia i rami decisionali, caratterizzanti tale procedura e quattro blocchi fondamentali, che saranno illustrati, nei particolari, nelle successive immagini.

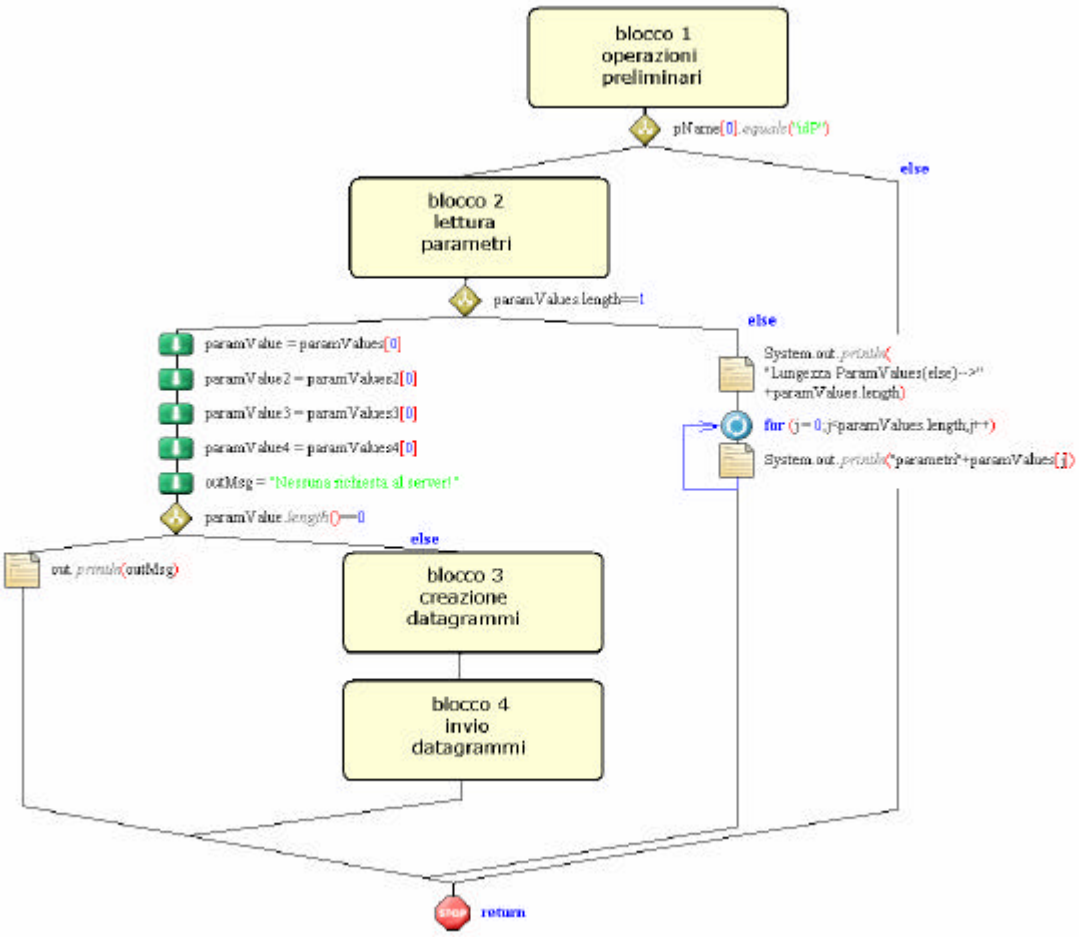


figura 3.39

In figura 3.40 viene illustrato il diagramma di flusso del metodo di inizializzazione della servlet.

Infatti nel ciclo di vita di una servlet viene innanzitutto chiamato il metodo init della superclasse, al quale viene passato l'oggetto ServletConfig che contiene i parametri della servlet e il riferimento a

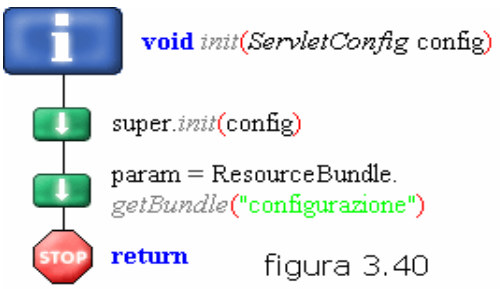


figura 3.40

ServletContext, che rappresenta il contesto in cui gira la servlet. Successivamente, viene letto il file di configurazione *"configurazione.properties".*

Nel blocco 1, in figura 3.41 vengono effettuate delle operazioni preliminari, quali istanziare le variabili che saranno utilizzate, la lettura dei parametri di configurazione dal file ".properties", come l'indirizzo IP della macchina che fornirà le forma d'onde lette (dove è installato il web server), la porta su cui il web server attende le richieste, il percorso che porta all'esecuzione della servlet in oggetto e quello per richiamare l'applet di visualizzazione e, infine, la lettura dei parametri nelle richieste tramite il metodo "getParameterNames", come già noto.

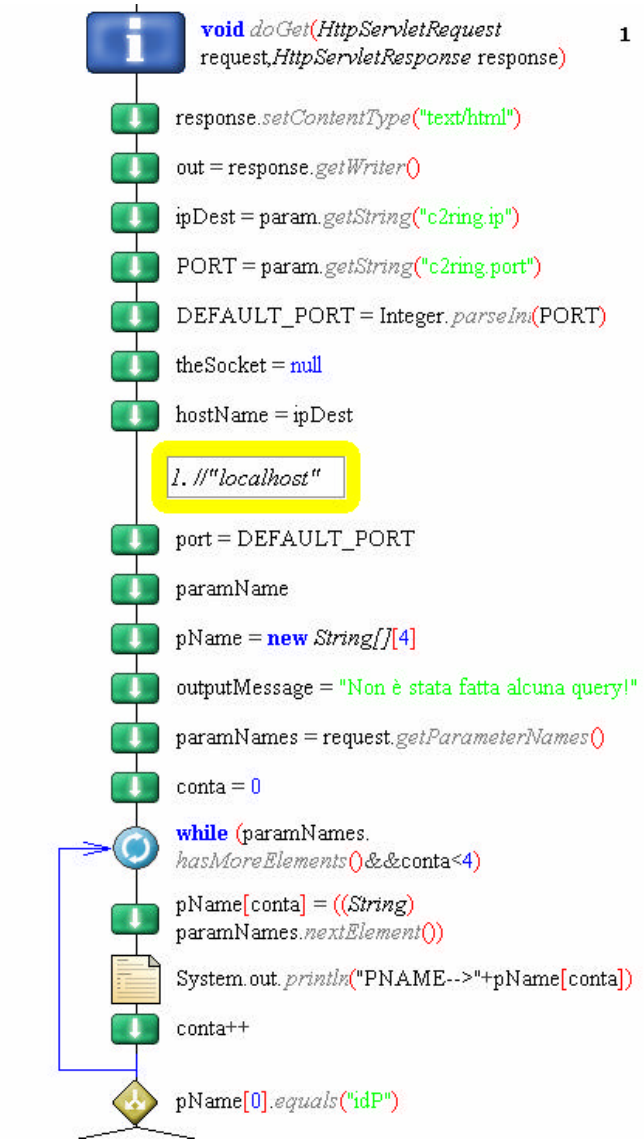


figura 3.41

Nel blocco 2, in figura 3.42, viene effettuata la lettura dei valori dei parametri nella richiesta, precedentemente letti.

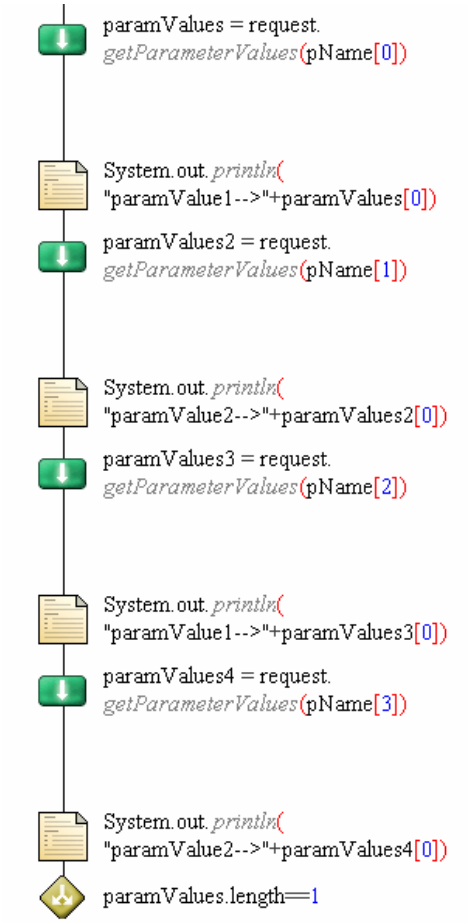


figura 3.42

Nel blocco 3, in figura 3.43, come precedentemente accennato, viene effettuata la "frammentazione" della stringa ricevuta, nelle sottostringhe contenenti le informazioni di picking, inviate dall' "applet dei sismogrammi". Una volta ottenute queste informazioni, vengono creati gli opportuni header, quello necessario al modulo coax2ring (di 6 byte) e quello necessario all'identificativo del tipo di pacchetto inviato, tra TYPE_PICK2K e TYPE_CODA2K.

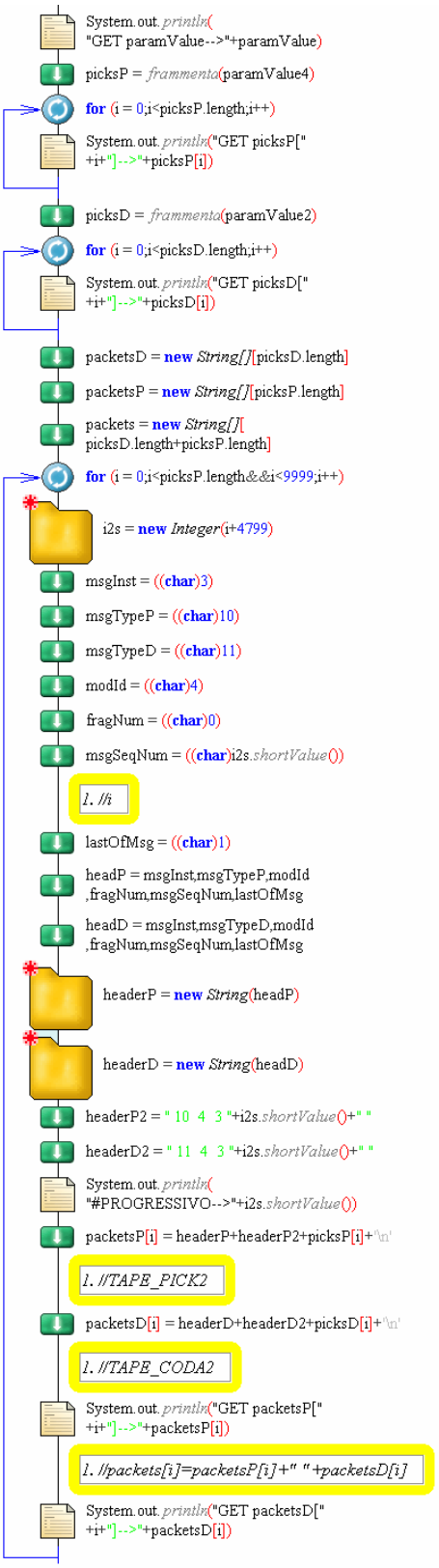
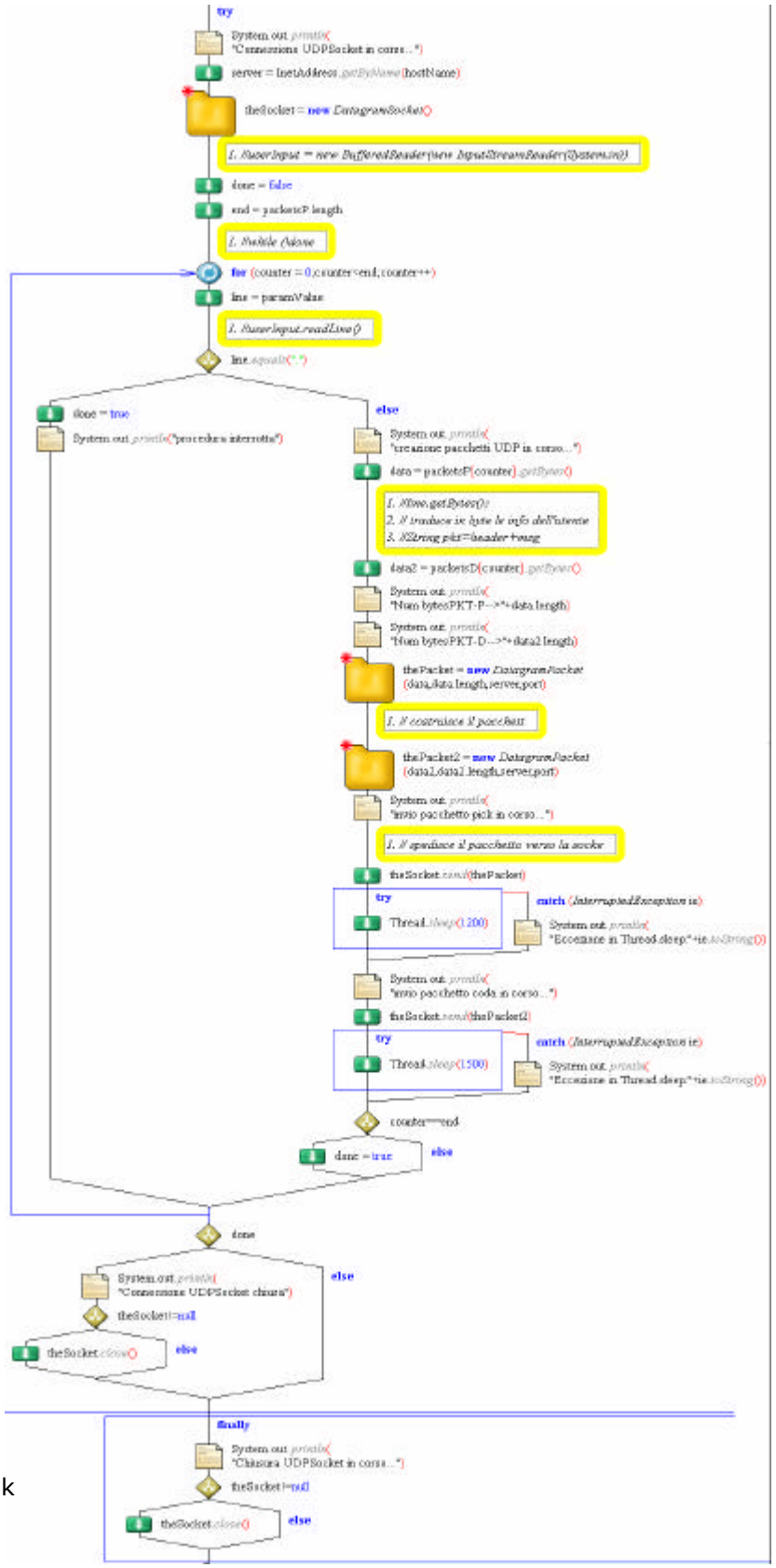


figura 3.43

Infine, in figura 3.44 sono illustrati i dettagli del blocco 4.


try
System.out.println("Connessione UDPSocket in corso...")
server = InetAddress.getByName(hostName)
theSocket = new DatagramSocket()
1. userInput = new BufferedReader(new InputStreamReader(System.in))
done = false
end = packetsP.length
1. Public (close
for (counter = 0;counter<end;counter++)
line = paramValue
1. userInput.readLine()
line.equals(".")
done = true
System.out.println("procedura interrotta")
else
System.out.println("creazione pacchetti UDP in corso...")
data = packetsP[counter].getBytes()
1. //low.getBytes()
2. // traduce in byte le info dell'utente
3. //String pkt=header+msg
data2 = packetsD[counter].getBytes()
System.out.println("Num bytesPKT-P-->"+data.length)
System.out.println("Num bytesPKT-D-->"+data2.length)
thePacket = new DatagramPacket (data,data.length,server,port)
1. // costruisce il pacchetto
thePacket2 = new DatagramPacket (data2,data2.length,server,port)
System.out.println("invio pacchetto pkt in corso...")
1. // spedisce il pacchetto verso la socket
theSocket.send(thePacket)
try
Thread.sleep(1200)
catch (InterruptedException ie)
System.out.println("Eccezione in Thread.sleep"+ie.toString())
System.out.println("invio pacchetto coda in corso...")
theSocket.send(thePacket2)
try
Thread.sleep(1500)
catch (InterruptedException ie)
System.out.println("Eccezione in Thread.sleep"+ie.toString())
counter==end
done = true
else
done
System.out.println("Connessione UDPSocket chiusa")
else
theSocket!=null
theSocket.close()
else
finally
System.out.println("Chiusura UDPSocket in corso...")
theSocket!=null
theSocket.close()
else

k

figura 3.44

In particolare, nel blocco 4, vengono effettuate le seguenti operazioni:

- apertura di una connessione socket UDP verso la macchina su cui risiede Earthworm, identificata da un proprio indirizzo IP e un numero di porta su cui riceve le richieste,
- traduzione in byte delle stringhe, ottenute nel blocco 3, contenenti le informazioni di picking;
- impacchettamento nei datagrammi UDP delle informazioni ottenute al passo precedente;
- invio, circa ogni secondo, dei datagrammi contenenti le informazioni di picking e di durata dell'evento, per ogni sismogramma dell'applet;
- chiusura della connessione socket.

Nel prossimo capitolo verranno illustrate le funzionalità e i dettagli implementativi degli applet, sviluppate per la realizzazione della console di comando e per la realizzazione di un'interfaccia grafica necessaria per il "rendering" dei segnali ottenuti dal Wave Server.

Come già accennato precedentemente, gli applet, in generale, sono limitati da questioni di sicurezza, a causa del fatto che vengono eseguiti sul sistema dell'utente. Se non fosse così, qualsiasi malintenzionato potrebbe scrivere un applet che cancelli i file dell'utente, raccolga informazioni private dal sistema e provochi altre violazioni di sicurezza. Quindi per poter permettere agli applet in oggetto di connettersi al WaveServer o di prelevare informazioni presenti in particolari files sulla macchina del web server, si è fatto in modo che le servlet svolgessero tutte quelle operazioni non consentite agli applet e lasciare a questi ultimi i compiti di attivare una particolare servlet in grado di compiere specifiche azioni (la console, ad esempio) e di prelevare informazioni ammissibili per la visualizzazione dei segnali .

Capitolo 4

Livello presentazione: gli applet

Gli Applet sono moduli software scritti in Java che vengono eseguiti all'interno di Web Browser e costituiscono un potente tool di supporto per la programmazione lato client.

Quando un utente con un browser compatibile con java carica una pagina Web nella quale è incluso un applet, il browser preleva l'applet da un server Web e lo esegue sul sistema dell'utente. Non è necessario un interprete Java a parte, poiché ce n'è uno all'interno del browser.

A causa del fatto che gli applet vengono eseguiti sul sistema di un utente Web, esistono rigide limitazioni alle operazioni che possono compiere. Se così non fosse, qualsiasi malintenzionato potrebbe scrivere un applet che cancelli i file dell'utente, raccolga informazioni private dal sistema e provochi altre violazioni della sicurezza. Quindi, un applet non può fare le seguenti operazioni:

- Leggere o scrivere file sul sistema dell'utente;
- Comunicare con un sito Internet diverso da quello su cui viene distribuita la pagina Web che include l'applet;
- Eseguire qualsiasi programma sul sistema di chi apre l'applet;
- Prelevare programmi registrati sul sistema dell'utente, quali programmi eseguibili e librerie condivise.

Salvo queste restrizioni, gli applet offrono diversi vantaggi, specialmente quando si costruiscono applicazioni client-server o altri tipi di applicazioni per il networking. Infatti un applet, avendo una piattaforma indipendente, non necessita di essere sviluppata in modo diverso a secondo del tipo di piattaforma su cui andrà a girare. Ciò è dovuto al fatto che l'installazione è automatica ogni volta che un utente carica la pagina Web che contiene l'applet. Nei tradizionali sistemi client-server, costruire ed installare una nuova versione di un software client è spesso un'operazione complessa.

Inoltre, grazie ai vincoli di sicurezza imposti ad un applet, non ci si "dovrebbe preoccupare" che il codice scaricato possa danneggiare il sistema.

Tutti gli applet sono sottoclassi della classe JApplet, contenuta nel package javax.swing, oppure della classe Applet, contenuta nel package java.applet. Ereditando le caratteristiche di una di queste due classi, gli applet presentano i seguenti comportamenti :

- Possibilità di funzionare come parte di un browser e gestire eventi quali il fatto che venga ricaricata la pagina Web;
- Possibilità di presentare un'interfaccia grafica utente e gestire gli input degli utenti.

Gli applet hanno metodi che vengono richiamati quando si verificano eventi specifici durante l'esecuzione. Ad esempio, ogni volta che è necessario visualizzare o aggiornare la finestra dell'applet, viene richiamato il metodo "paint()".

I metodi più importanti della classe Applet, che vengono ridefiniti nella scrittura di un applet, sono:

- *Inizializzazione*: "init()". Questo metodo viene eseguito in fase di caricamento dell'applet e può includere la creazione degli oggetti necessari per l'applet, la definizione di uno stato iniziale, il caricamento di immagini e tipi di carattere o l'impostazione di parametri.
- *Avvio*: "start()". E' il metodo per l'avvio dell'applet dopo l'inizializzazione o dopo un arresto e può aver luogo più volte nel ciclo di vita dell'applet.
- *Arresto*: "stop()". L'arresto ha luogo quando l'utente abbandona la pagina che contiene l'applet in esecuzione, oppure quando si richiama il metodo stop().
- *Distruzione*: "destroy()". Il metodo destroy() consente a un applet di "resettare" l'ambiente appena prima di essere scaricato dalla memoria o dall'uscita da browser. Ad esempio, si potrebbe invocare tale metodo per rilasciare gli oggetti ancora in esecuzione. Comunque, Java possiede un meccanismo automatico di raccolta degli elementi non più utilizzati all'interno della memoria. Questo meccanismo recupera la memoria occupata dalle risorse dopo che il programma ha terminato di utilizzarle, perciò normalmente non vi è necessità di utilizzare un tale metodo.
- *Disegno*: paint()". Per disegno si intende qui il modo in cui un applet traccia sullo schermo un testo, una linea, uno sfondo a colori o un'immagine ed è questa un'operazione che può aver luogo molte volte nel ciclo di vita di un applet. Per visualizzare qualsiasi cosa bisogna ridefinire il metodo paint() della sottoclasse Applet. Questo metodo accetta come argomento un'istanza della classe Graphics, la quale viene creata e passata a paint() dal browser.

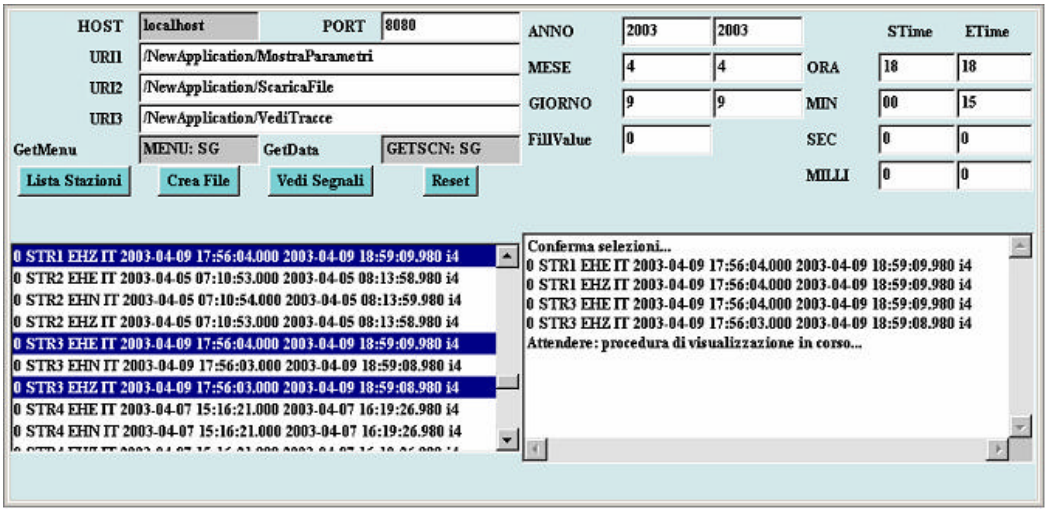
Nei paragrafi, seguenti, saranno descritti gli applet necessari per l'implementazione delle due interfacce grafiche introdotte nei capitoli

precedenti, una per la realizzazione della console di comando e l'altra per la visualizzazione dei sismogrammi a display. In particolare, grazie alla console di comando si possono eseguire le seguenti operazioni sul wave-server:

- Ottenere una lista delle stazioni attive nella rete di acquisizione;
- Ottenere i sismogrammi di alcune di queste stazioni, a partire da un certo istante e specificando la durata, a seconda delle esigenze dell'utente. Su questi segnali si possono, poi, effettuare operazioni di zoom, rimozione della componente continua, normalizzazione a fondo scala, "spanning temporale", rilevazione delle informazioni di picking e durata di un eventuale evento e invio di queste ultime ai moduli lato server, descritti nel capitolo 3, sviluppati per la localizzazione dell'evento stesso.

Questi applet sono realizzati da moduli software raggruppati nel package "TraceView".

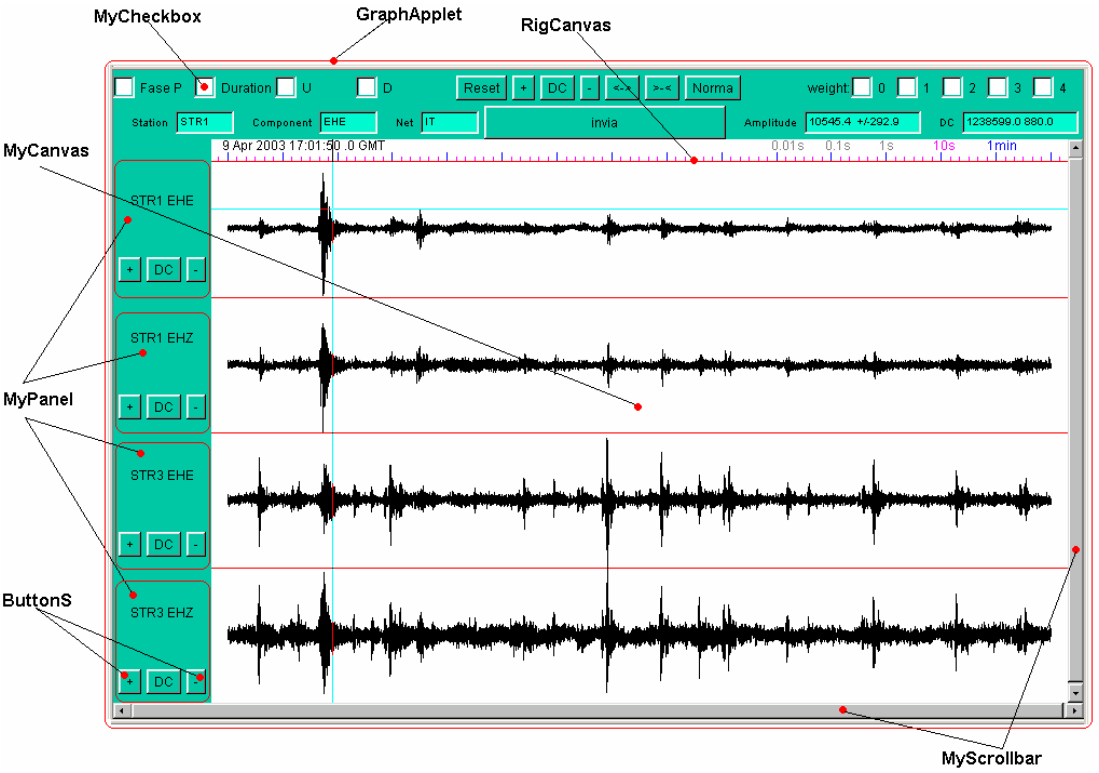
Di seguito sono riportate le immagini della console e dell'applet dei sismogrammi nel caso siano state visualizzate delle tracce. In particolare, in quest'ultima sono evidenziati tutti i componenti precedentemente illustrati.



Console di comando

In figura, nel riquadro a sinistra è evidenziata parte delle stazioni presenti nella lista delle stazioni attive del Wave Server, mentre il riquadro a destra rappresenta un area di testo in cui saranno visualizzati messaggi sul corretto funzionamento del sistema. Sono, inoltre, evidenti tutti i componenti introdotti, nella discussione sull'applet SendPost2, nel paragrafo 4.2.

Nella prossima figura viene illustrata l'applet GraphApplet ed evidenziati tutti i componenti finora discussi.



GraphApplet

Tale applet è caratterizzata da un array MX1, con M pari al numero di sismogrammi da visualizzare, tale che ogni riga è rappresentato da un "pannello grafico". Ogni pannello permette la visualizzazione di un sismogramma e su questo delle operazioni di zoom in positivo o in negativo e l'eliminazione di un eventuale componente continua presente nella forma d'onda. Le stesse operazioni possono essere effettuate contemporaneamente su tutti i sismogrammi, agendo sul pannello principale in alto. Su questo, inoltre, è anche possibile selezionare il tipo di operazione di picking, se l'estrazione dei tempi di arrivo dell'evento o la durata dell'evento, agendo su una delle due checkbox: "faseP" o "duration". In seguito a tali operazioni preliminari, è possibile avviare la procedura di localizzazione automatica, tipica di Earthworm, inviando le informazioni di picking estratte ad Earthworm, semplicemente agendo sul tasto "invia" del pannello principale.

4.1 Diagramma delle classi

Nel presente paragrafo verranno descritte le relazioni e le dipendenze intrinseche tra le varie classi del package traceView.
In figura 4.1 è illustrato il diagramma delle classi degli applet "SendPost2.java" e "GraphApplet.java", nonché i metodi in essi implementati.

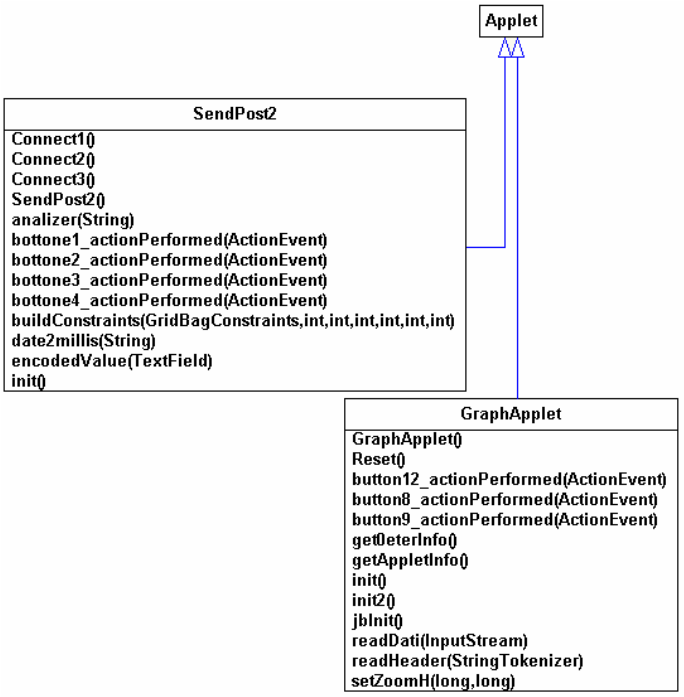


figura 4.1

Da tale figura è evidente come questi due oggetti siano sottoclassi della classe principale Applet, del package java.applet. All'interno di ognuno di questi due oggetti, sono realizzate altre classi, con metodi propri, necessarie per dare particolari funzionalità agli oggetti forniti da Java, nel package java.awt, come ad esempio un pannello o un bottone.
In figura 4.2 è illustrato il diagramma delle classi che estendono la classe Button e alle quali è associato un "ascoltatore di eventi". Tale ascoltatore rileva le azioni che un utente compie su un componente, come ad esempio un clic di mouse.

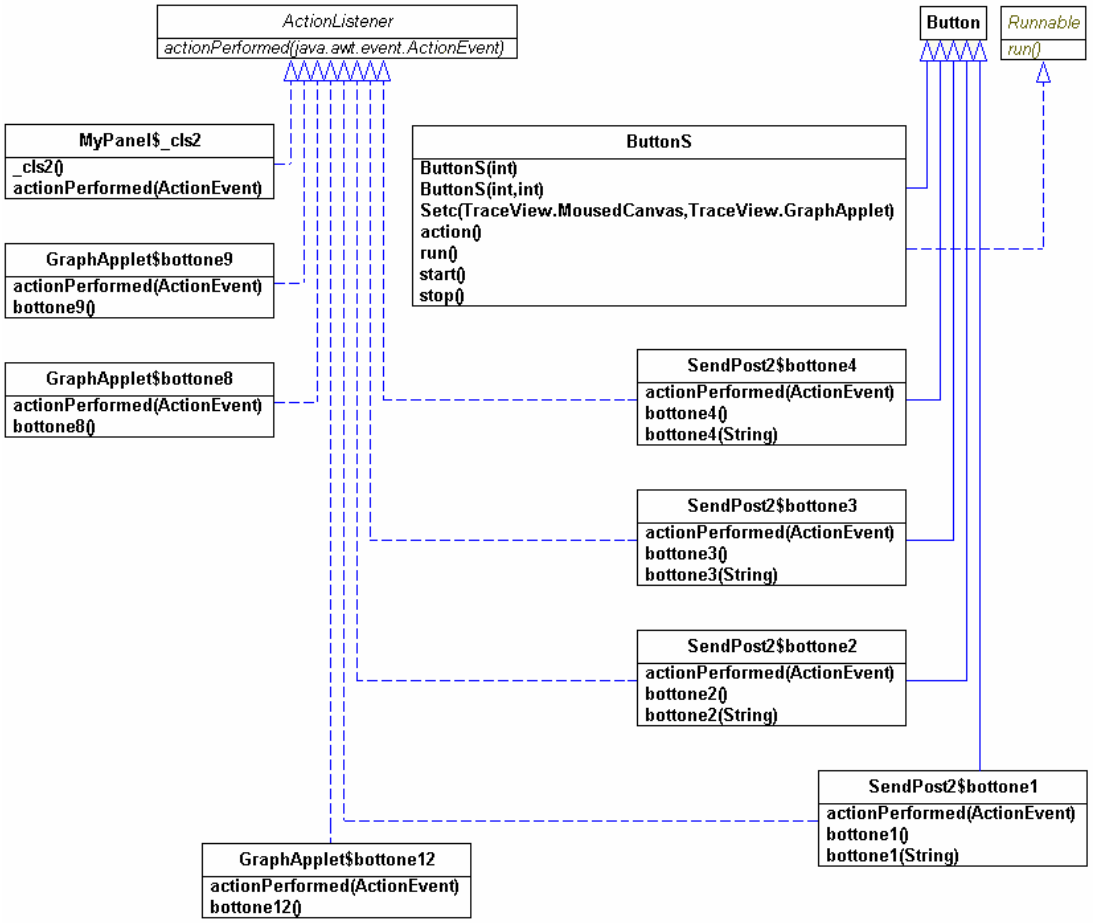


figura 4.2

Si può notare che l'oggetto "ButtonS.java" è una sottoclasse della classe java.awt.Button ed implementa l'interfaccia "Runnable" del package java.lang, in modo da poter utilizzare i "thread".

Un thread è una parte di un programma predisposta per essere eseguita autonomamente. Tale possibilità è chiamata anche *multitasking*, dato che il programma può trattare più compiti contemporaneamente. Gestendo, ad esempio, il carico di lavoro di un'animazione all'interno di un thread, si libera il resto del programma consentendo a quest'ultimo di gestire altre azioni.

Gli oggetti, appartenenti a GraphApplet, implementano l'interfaccia "ActionListener" per dare le funzionalità necessarie, quando viene rilevata un azione dell'utente. Questi oggetti saranno usati come ascoltatori di eventi sui bottoni per lo zoom, la rimozione della componente continua, la normalizzazione a fondo scala e il reset.

Gli oggetti, appartenenti a SendPost2, invece, estendono le funzionalità della classe Button, ma implementano direttamente l'ascoltatore di eventi.

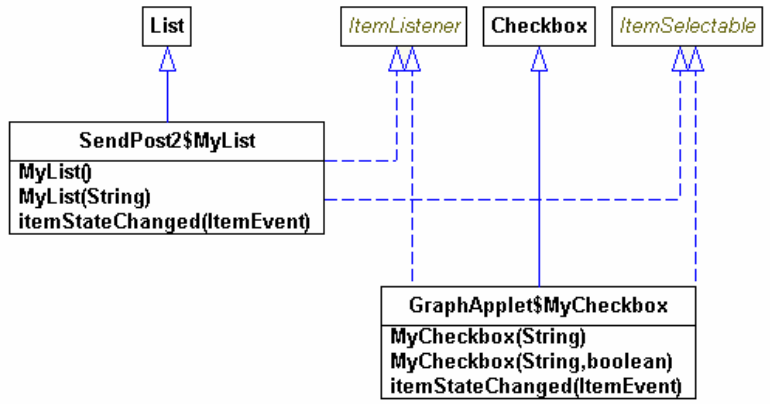


figura 4.3

In figura 4.3 è illustrato il diagramma delle classi relative agli oggetti "List" e "Checkbox".

Il primo oggetto viene utilizzato per creare la lista delle stazioni e l'ascoltatore di eventi rileva tutti gli elementi che l'utente seleziona. Il secondo oggetto viene utilizzato per abilitare delle funzionalità ed inibirne altre, come nel caso in cui si stiano selezionando le informazioni di picking o di durata. In tal modo queste operazioni possono essere effettuate in maniera distinta o simultanea.

In figura 4.4 è illustrato il diagramma delle classi RigCanvas e MyCanvas.

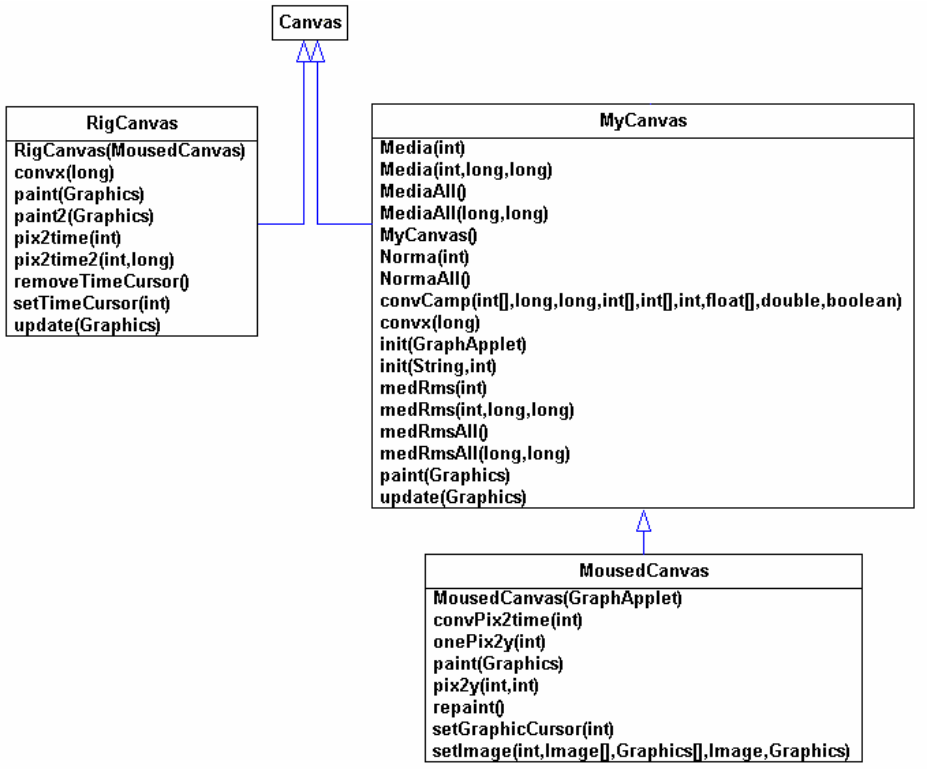


figura 4.4

Queste classi caratterizzano le funzionalità grafiche dell'applet dei sismogrammi, nel senso che grazie a queste è possibile visualizzare a display i sismogrammi selezionati dall'utente. Esse sono sottoclassi della classe Canvas. In più l'oggetto MousedCanvas estende la classe MyCanvas per fornire quest'ultima degli eventi di mouse, generati da un click, dal puntamento e dall'abbandono di un'area del componente, e degli eventi di movimento del mouse, utilizzati per tracciare tutti gli spostamenti del mouse sul componente stesso.

In particolare, in figura 4.5, è illustrato il diagramma delle classi che vengono

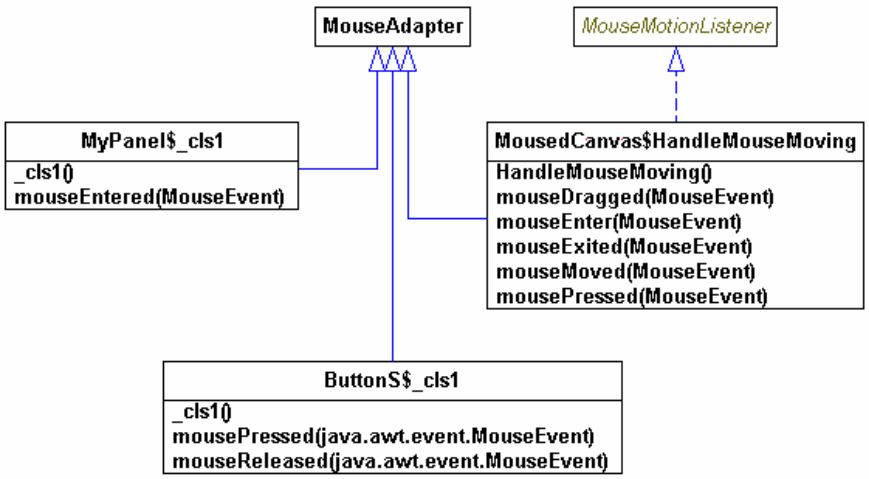


figura 4.5

fornite delle funzionalità di "evento mouse". Oltre a MousedCanvas, la classe _cls1 di MyPanel è dotata di un ascoltatore sul mouse per rilevare l'ingresso del puntatore in un'area dedicata ad un particolare sismogramma e indicarne sul pannello le informazioni che identificano il sismogramma stesso, come la terna SCN (*Station Component Network*). La classe _cls1 di ButtonS invece rileva gli eventi di pressione e rilascio dei pulsanti del mouse su tale bottone, per compiere delle azioni impostate nei metodi mousePressed() e mouseReleased(), rispettivamente.

In figura 4.6 è illustrato il diagramma delle classi grazie alle quali si può dotare l'applet dei sismogrammi delle funzionalità di scrollbar, estendendo quelle della classe Scrollbar, fornita da Java.

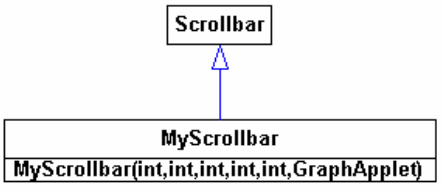


figura 4.6

In figura 4.7 è illustrato il diagramma delle classi che implementano l'interfaccia

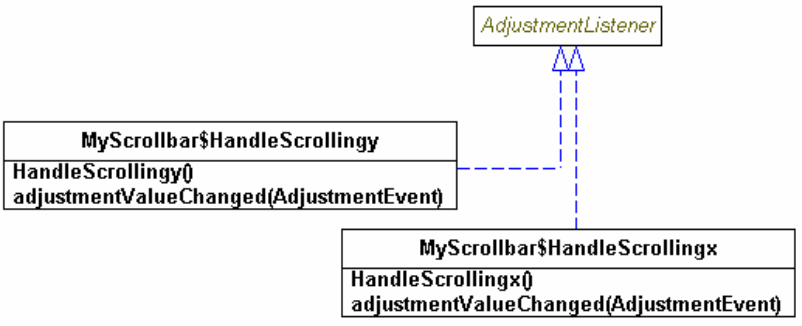


figura 4.7

AdjustmentListener, ascoltatore di eventi che riceve le azioni dell'utente sulla barra di scrolling verticale o orizzontale.

Infine, in figura 4.8, è illustrato il diagramma delle classi che estendono le caratteristiche dell'oggetto Panel, superclasse di MyPanel, LabeledTextField e _cls1 di GraphApplet.

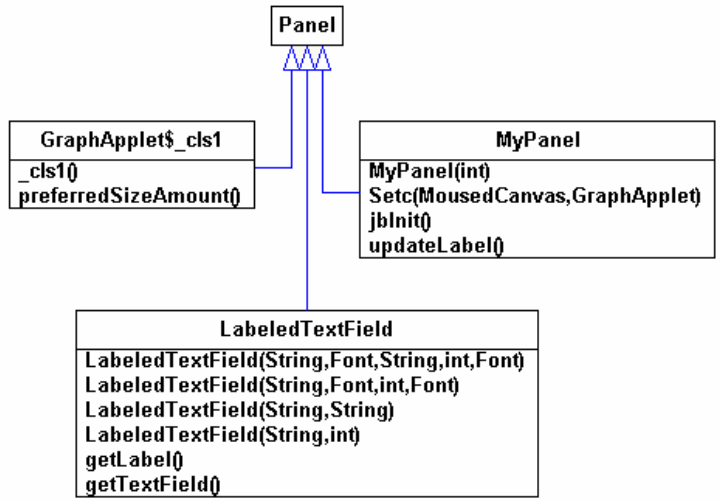


figura 4.8

4.2 SendPost2.java

Nel presente paragrafo viene data una descrizione delle funzionalità dell'applet "SendPost2.java" nel tempo.

In figura 4.9 è illustrato il diagramma temporale del metodo init().

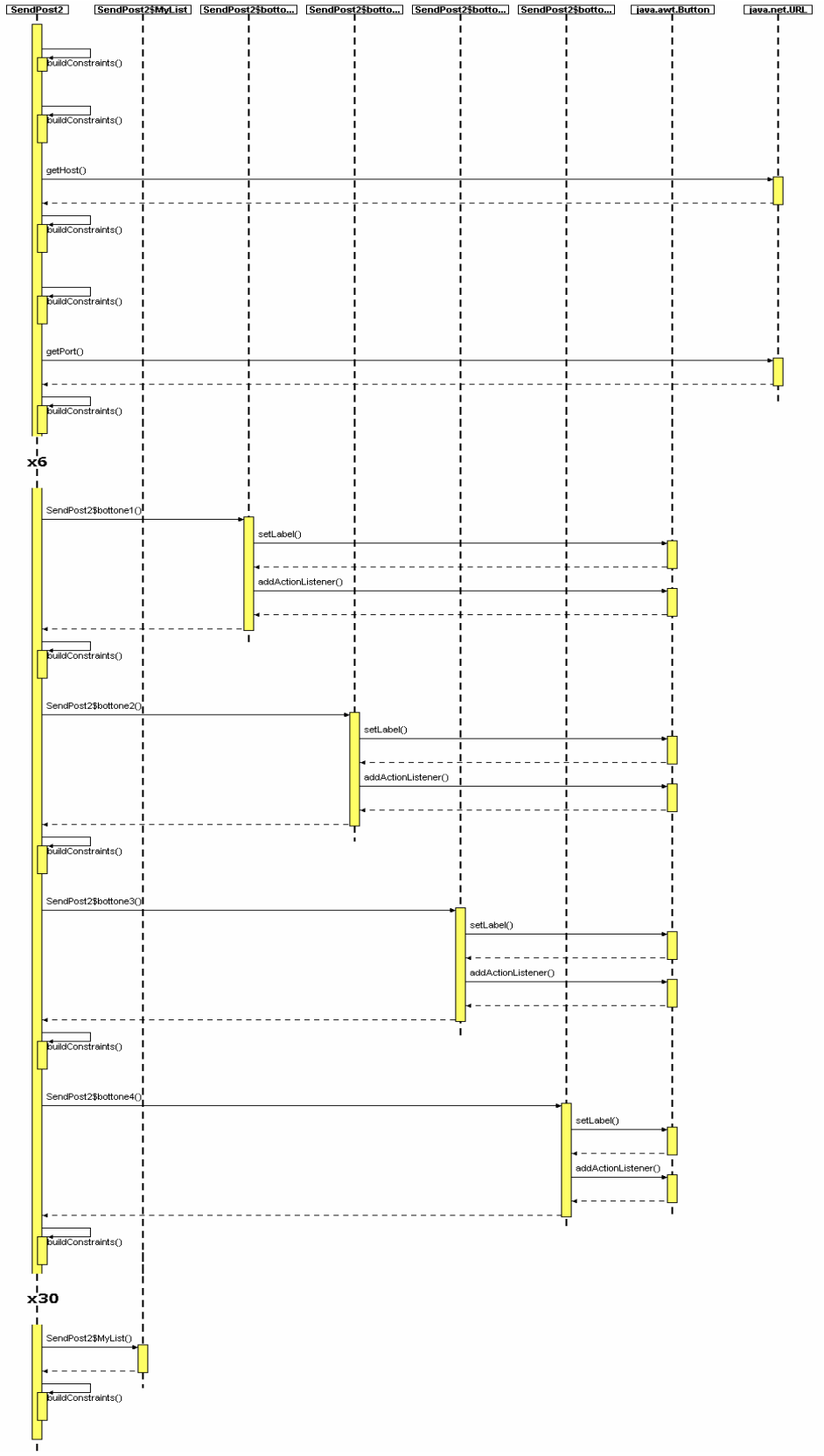

SendPost2
SendPost2$MyList
SendPost2$botto...
SendPost2$botto...
SendPost2$botto...
SendPost2$botto...
java.awt.Button
java.net.URL
buildConstraints()
buildConstraints()
getHost()
buildConstraints()
buildConstraints()
getPort()
buildConstraints()
x6
SendPost2$bottone1()
setLabel()
addActionListener()
buildConstraints()
SendPost2$bottone2()
setLabel()
addActionListener()
buildConstraints()
SendPost2$bottone3()
setLabel()
addActionListener()
buildConstraints()
SendPost2$bottone4()
setLabel()
addActionListener()
buildConstraints()
x30
SendPost2$MyList()
buildConstraints()

figura 4.9

Questo metodo, oltre a prelevare informazioni, quali indirizzo IP del computer su cui gira il Web server e la porta su cui quest'ultimo riceve e invia informazioni, compie le seguenti operazioni:

- avviare il metodo buildConstraints(), grazie al quale è possibile dare un layout personalizzato dell'applet, in modo da forzare manualmente l'inserimento di oggetti come bottoni o campi di testo, senza dover usare dei layout predefiniti di Java;
- istanziare oggetti, quali "bottoni";
- associare ascoltatori di eventi sui bottoni.

Le ultime due operazioni sono illustrate in figura 4.10.

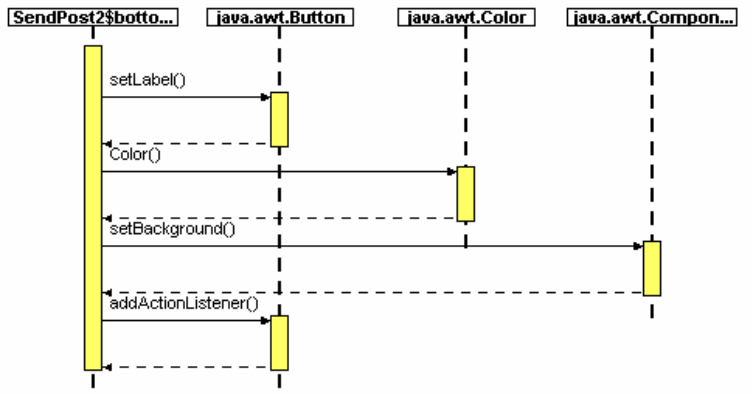


figura 4.10

In particolare, vengono istanziati quattro oggetti "bottone", che estendono il componente Button, ognuno dei quali con un tipo di azione diversa da compiere, nel caso sia rilevato un evento d'utente.

Il primo bottone, nominato "Lista Stazioni", è quello su cui agire per ottenere una lista delle stazioni attive; il secondo bottone è "crea file", grazie al quale è possibile scaricare i sismogrammi, digitalizzati, in un file, senza visualizzarne il contenuto sul display; il terzo bottone è "vedi segnali". Agendo su questo componente si crea un file contenente i campioni dei sismogrammi desiderati, per permettere un'analisi off-line del contenuto, e si mostrano a video le tracce; il quarto bottone, "reset", ha lo scopo di riportare le condizioni della console dei comandi allo stato iniziale, come ad esempio deselezionare tutte le stazioni in lista, se queste fossero state precedentemente selezionate.

Nelle figure 4.11, 4.12, 4.13, 4.14 vengono illustrati i dettagli delle operazioni che vengono compiute, agendo su uno di questi bottoni.

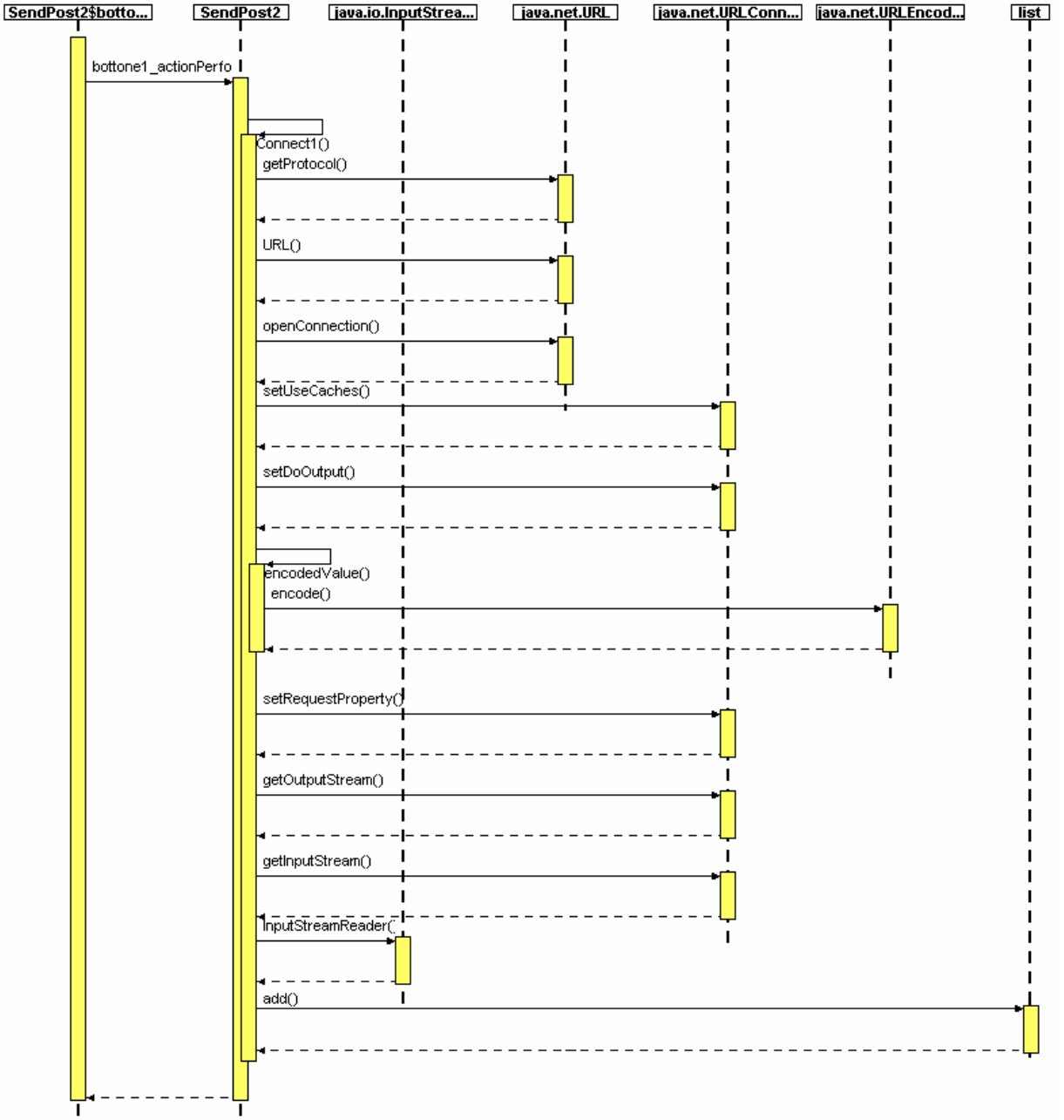


figura 4.11: ”crea lista”

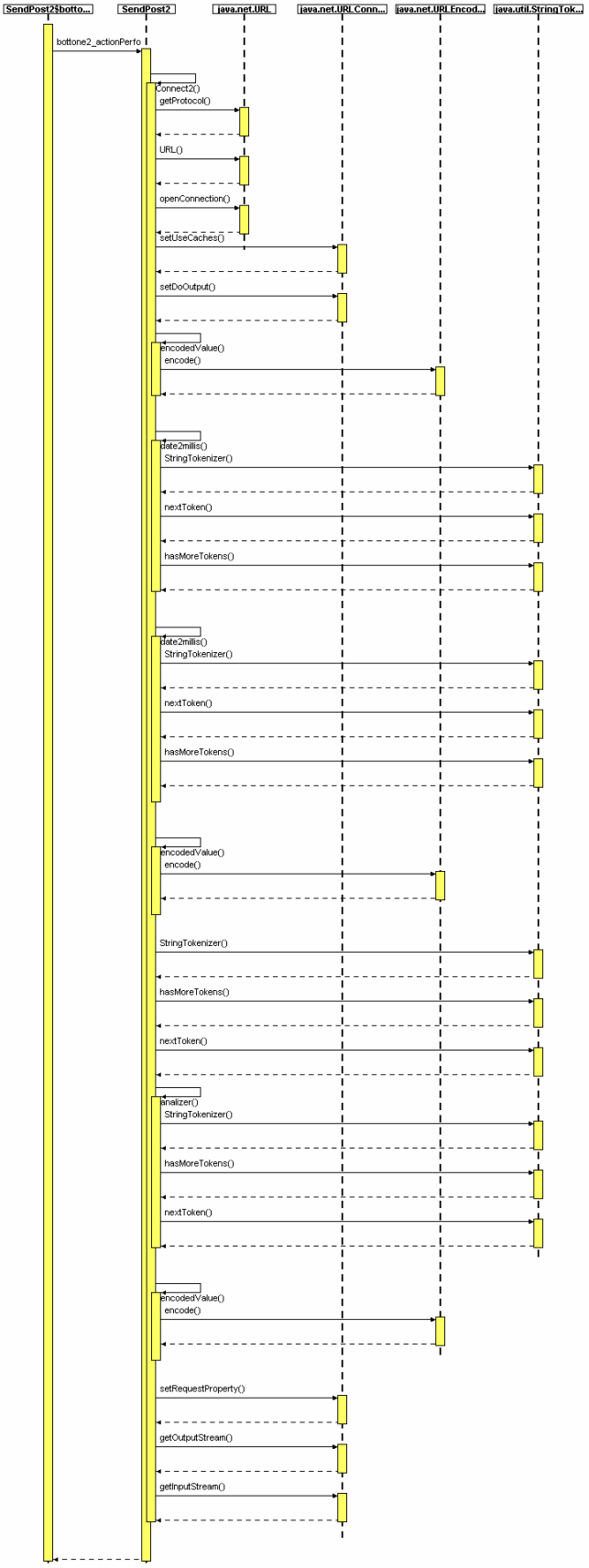


figura 4.12: "crea file"

figura 4.13:"vedi segnali"

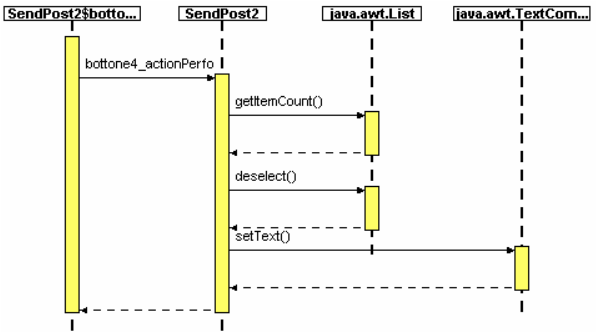


figura 4.14: "reset"

Come si può notare, l'azione dell'utente sul bottone viene tradotta nell'invio di informazioni verso un particolare URL, che, a seconda dei casi, è associato ad una particolare servlet (tra quelle precedentemente discusse). Queste operazioni di invio richieste e ricezione risposte vengono effettuate dai metodi Connect1(), Connect2(), Connect3().

Nel metodo Connect1(), una volta ricevute le informazioni dalla servlet "ShowParametersNew.java", viene creata la lista delle stazioni tramite l'oggetto list, istanza di "MyList", che a sua volta estende le funzionalità della classe java.awt.List. In figura 4.15, sono illustrati i dettagli su come viene creato l'oggetto MyList e come su questo viene imposto un ascoltatore di eventi, per la gestione delle selezioni degli "item". I dettagli che implementano tale funzionalità sono illustrati in figura 4.16.

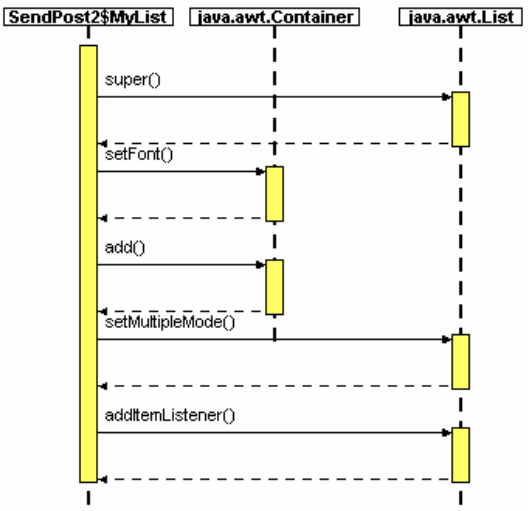


figura 4.15

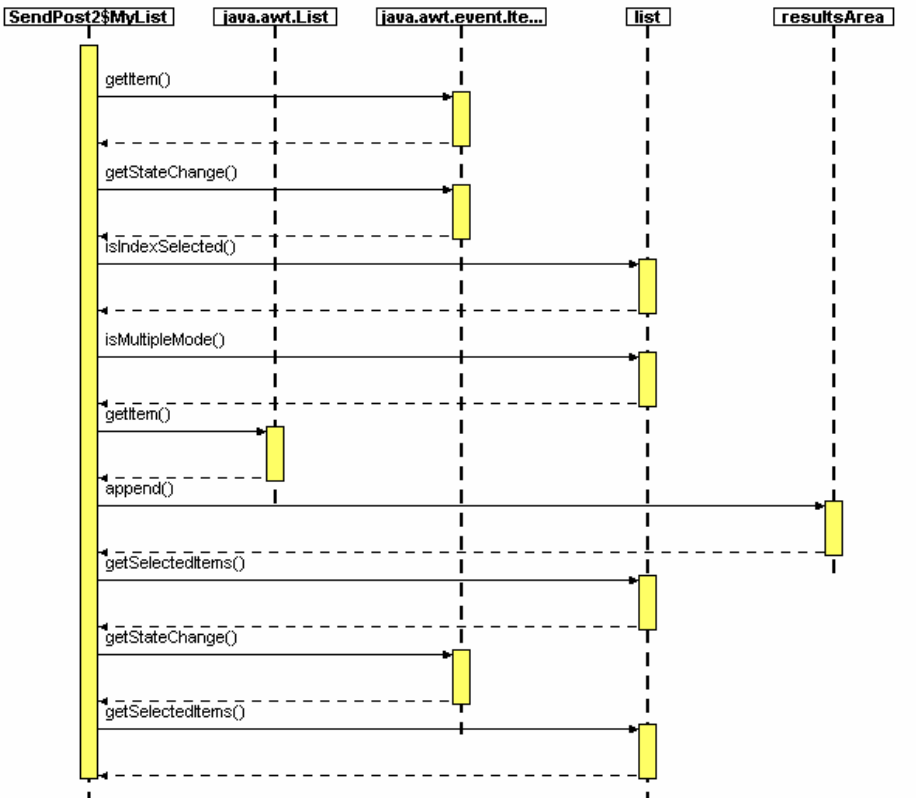


figura 4.16

Da questa si può notare come gli item, una volta selezionati, vengono visualizzati in un'area di testo, inserita con lo scopo di visualizzare messaggi di output per il controllo del buon funzionamento dei collegamenti o semplicemente visualizzare ciò che è stato selezionato.

Il metodo Connect2() invia le informazioni necessarie per richiedere le forme d'onda al Wave Server, tramite la servlet WaveDownloadNew.java, crea una sorta di puntatore temporaneo al file creato dalla servlet stessa, contenente i sismogrammi in forma digitale, e nel caso in cui l'utente abbia scelto di visualizzare le tracce, passa tale puntatore a Connect3(). Infatti quest'ultimo metodo, tramite la servlet MasterServ.java passa l'URL al file creato, in modo che quest'ultimo

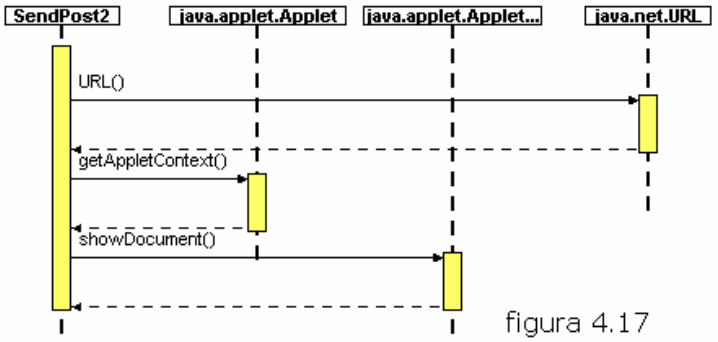


figura 4.17

possa essere letto dall'applet di visualizzazione. In figura 4.17 è illustrato il diagramma temporale che caratterizza il metodo Connect3(), mentre la figura 4.18 mostra le caratteristiche temporali del metodo date2millis, usato per convertire lo "start time" e l' "end time" in millisecondi a partire dall'istante standard 00:00:00 GMT del 01-01-1970.

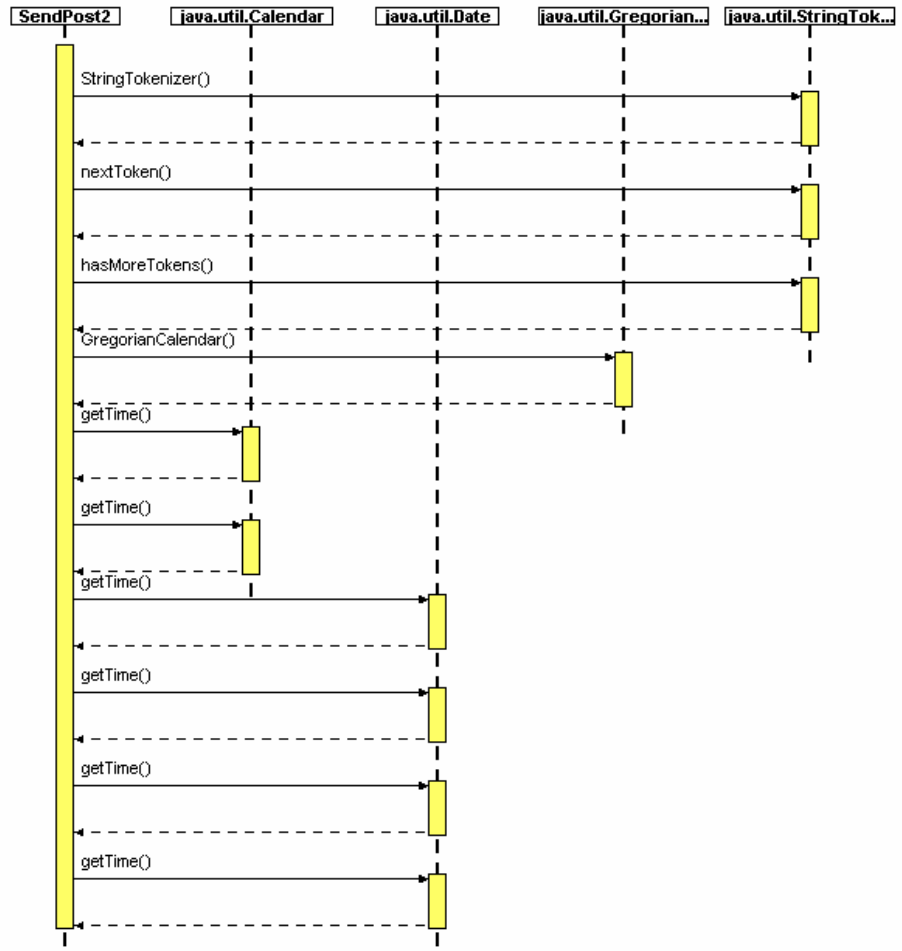


figura 4.18

4.3 GraphApplet.java

Nel presente paragrafo viene data una descrizione delle funzionalità dell'applet "GraphApplet.java" nel tempo e si illustra come da questa si attivino tutti i moduli necessari alla visualizzazione dei sismogrammi. In figura 4.19 è illustrato il diagramma temporale del metodo costruttore di GraphApplet. Qui vengono istanziate le classi ButtonS, e _cls1 di ButtonS, MyCheckbox e _cls1, che saranno descritte in seguito.

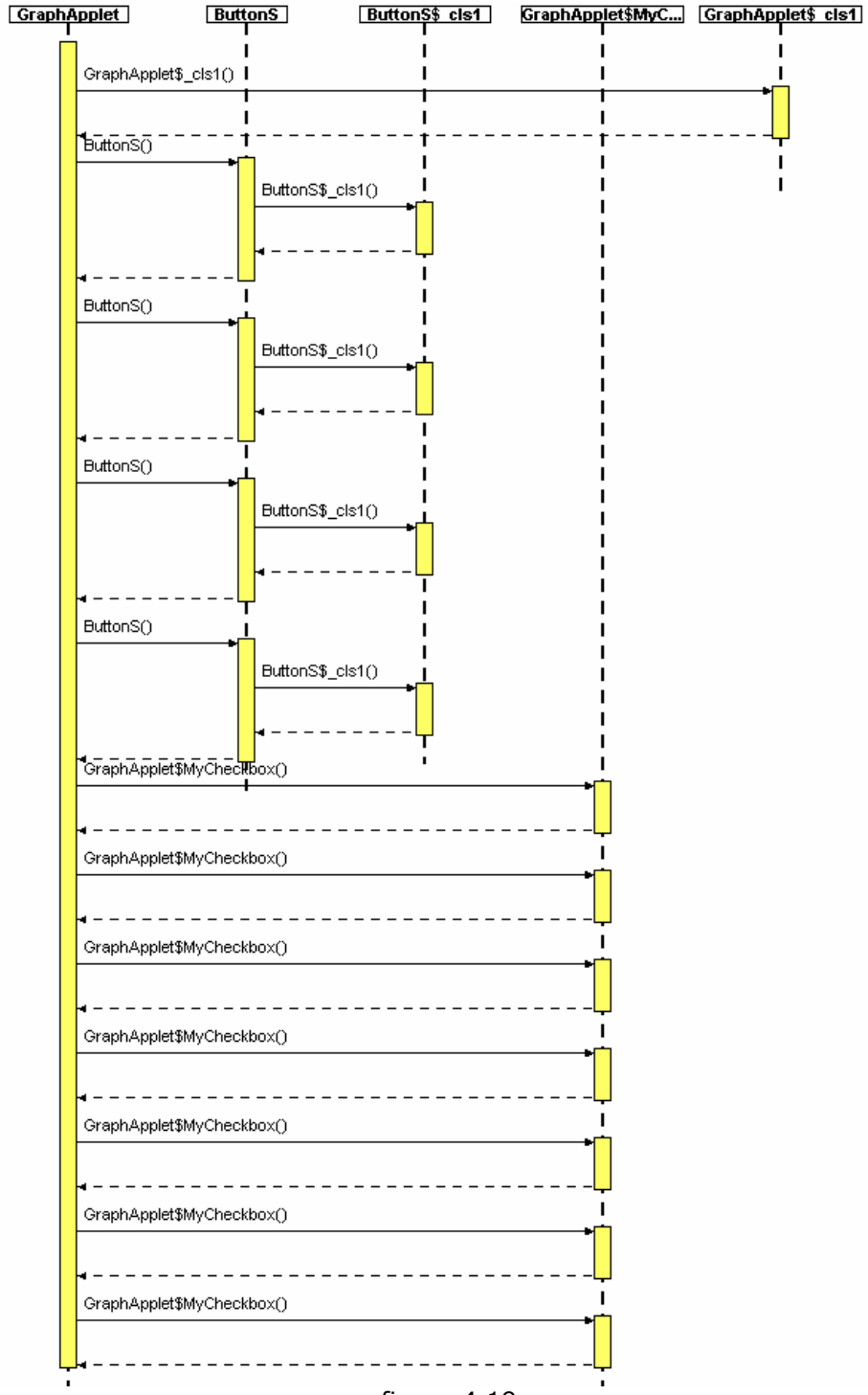


figura 4.19

In figura 4.20 e 4.21 sono illustrati i diagrammi temporali dei metodi jbinit() e init2(), richiamati dal metodo init().

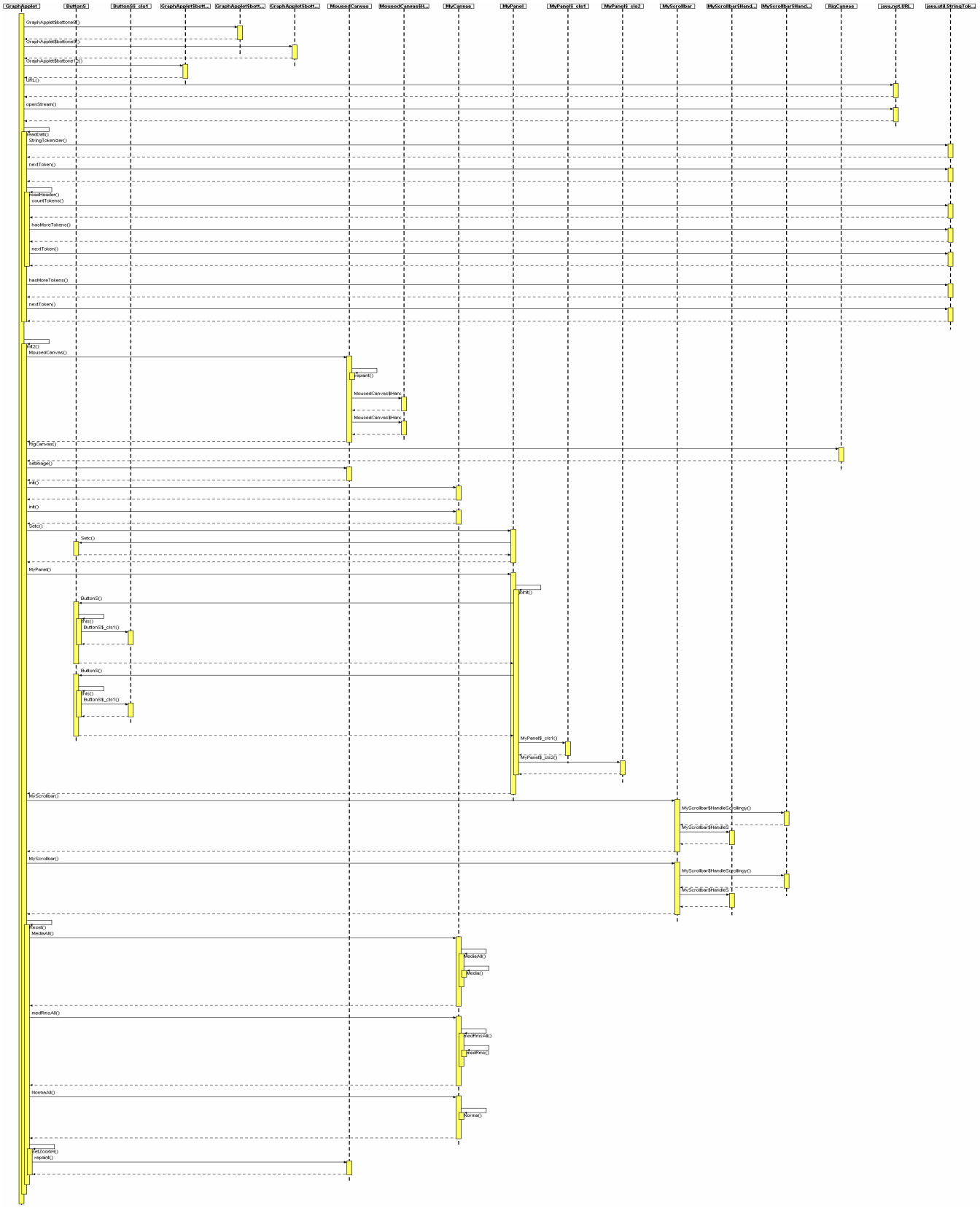

GraphApplet
ButtonS
ButtonS$_cls1
GraphApplet$bott...
GraphApplet$bott...
GraphApplet$bott...
MousedCanvas
MousedCanvas$Ha...
MyCanvas
MyPanel
MyPanel$_cls1
MyPanel$_cls2
MyScrollbar
MyScrollbar$Hand...
MyScrollbar$Hand...
RigCanvas
java.net.URL
java.util.StringTok...
GraphApplet$bottons$()
GraphApplet$bottons$()
GraphApplet$bottons$_2()
URL()
openStream()
readDat()
StringTokenizer()
nextToken()
readHeader()
countTokens()
hasMoreTokens()
nextToken()
hasMoreTokens()
nextToken()
M2()
MousedCanvas()
repaint()
MousedCanvas$Hank
MousedCanvas$Hank
RigCanvas()
setImage()
Init()
init()
SetCo()
SetCo()
MyPanel()
$Init()
ButtonS()
this()
ButtonS$_cls1()
ButtonS()
this()
ButtonS$_cls1()
MyPanel$_cls1()
MyPanel$_cls2()
MyScrollbar()
MyScrollbar$HandleScrollingY()
MyScrollbar$Handle$
MyScrollbar()
MyScrollbar$HandleScrollingY()
MyScrollbar$Handle$
Reset()
MediaAll()
MediaAll()
Media()
readRmsAll()
readRmsAll()
readRms()
NormaAll()
Norma()
getZoomIn()
repaint()

figura 4.20

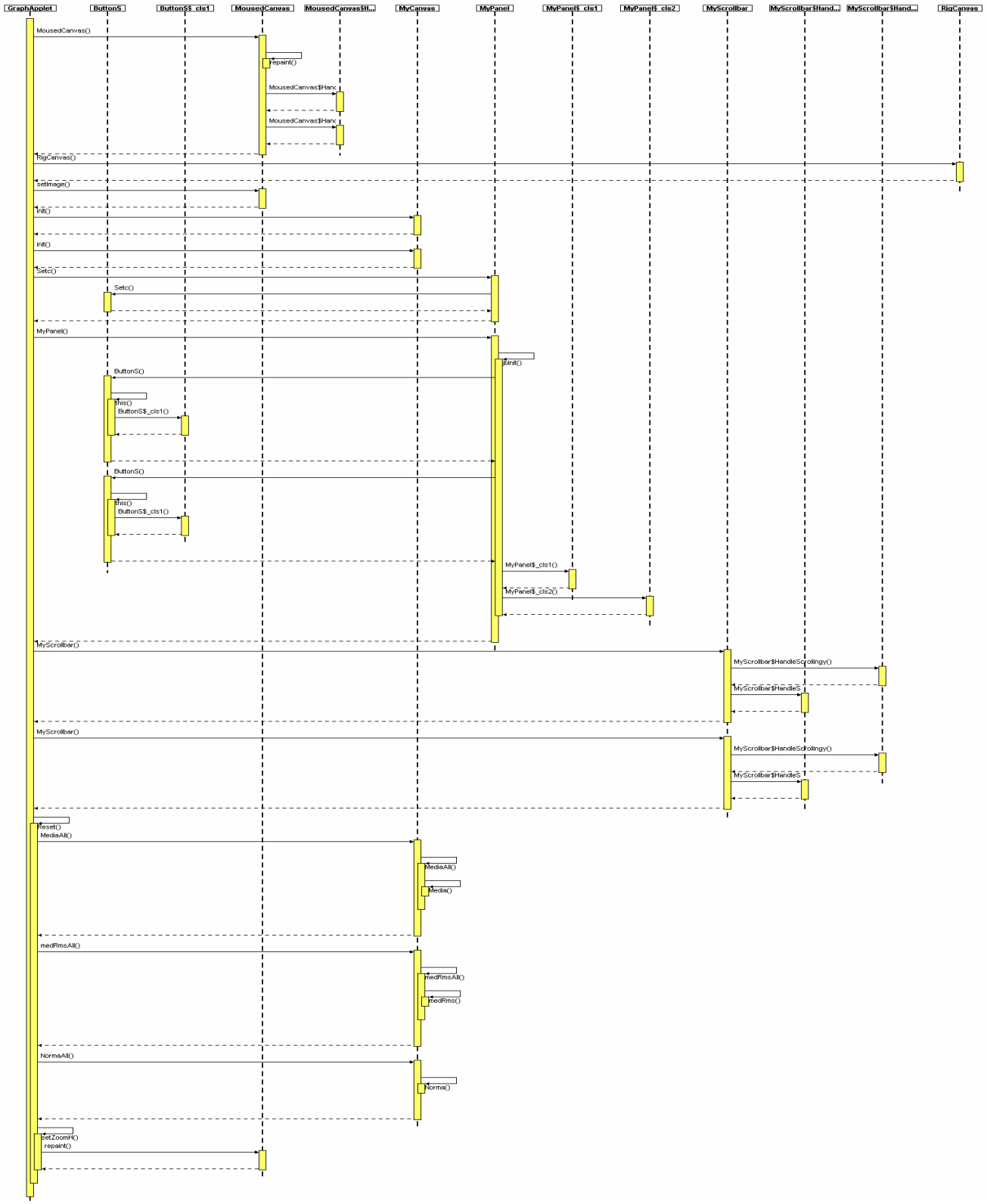

GraphApplet
ButtonS
ButtonS$_cls1
MousedCanvas
MousedCanvas$H...
MyCanvas
MyPanel
MyPanel$_cls1
MyPanel$_cls2
MyScrollbar
MyScrollbar$Hand...
MyScrollbar$Hand...
RigCanvas
MousedCanvas()
repaint()
MousedCanvas$Hanc
MousedCanvas$Hanc
RigCanvas()
setImage()
init()
init()
SetC()
SetC()
MyPanel()
$init()
ButtonS()
this()
ButtonS$_cls1()
ButtonS()
this()
ButtonS$_cls1()
MyPanel$_cls1()
MyPanel$_cls2()
MyScrollbar()
MyScrollbar$HandleScrollingy()
MyScrollbar$HandleS
MyScrollbar()
MyScrollbar$HandleScrollingy()
MyScrollbar$HandleS
Reset()
MediaAll()
MediaAll()
Media()
medRmsAll()
medRmsAll()
medRms()
NormaAll()
Norma()
SetZoomH()
repaint()

figura 4.21

In particolare, nel metodo jbinit(), una volta istanziati tutti i componenti grafici necessari, viene aperto uno stream di dati dall'URL passato dalla servlet MasterServ.java e che punta al file contenente i campioni dei sismogrammi che si vogliono visualizzare.
Allo scopo di leggere le informazioni dallo stream viene creato il metodo readDati(). Esso crea un array (NxM), con N il numero di sismogrammi da visualizzare e M il numero di campioni dell'N-esimo sismogramma. Da notare che, poiché tutti i segnali vengono campionati con uno stesso passo di campionamento **R** (100 c/s) e poiché tutti i sismogrammi vengono richiesti tra uno stesso start time e uno stesso end time, M, il numero di campioni dell'N-esima stazione, è uguale per tutti i sismogrammi. Infatti se

$$\Delta t = (endTime - StartTime)$$

e R è il passo di campionamento, uguale per tutti i segnali, il numero di campioni C sarà dato dalla banale equazione:

$$C = \Delta t * R$$

Il metodo init2(), istanzia gli oggetti MyCanvas, RigCanvas, MyPanel e MyScrollbar, necessari per il "rendering" dei sismogrammi, che saranno illustrati nei prossimi paragrafi.
In figura 4.22 è illustrato il diagramma temporale che caratterizza "readDati()".
Da questo si può notare la chiamata al metodo readHeader(), tramite il quale si crea una matrice (Nx1), la cui riga N-esima conterrà l'header dell'N-esimo sismogramma, e il cui diagramma temporale è illustrato in figura 4.23.

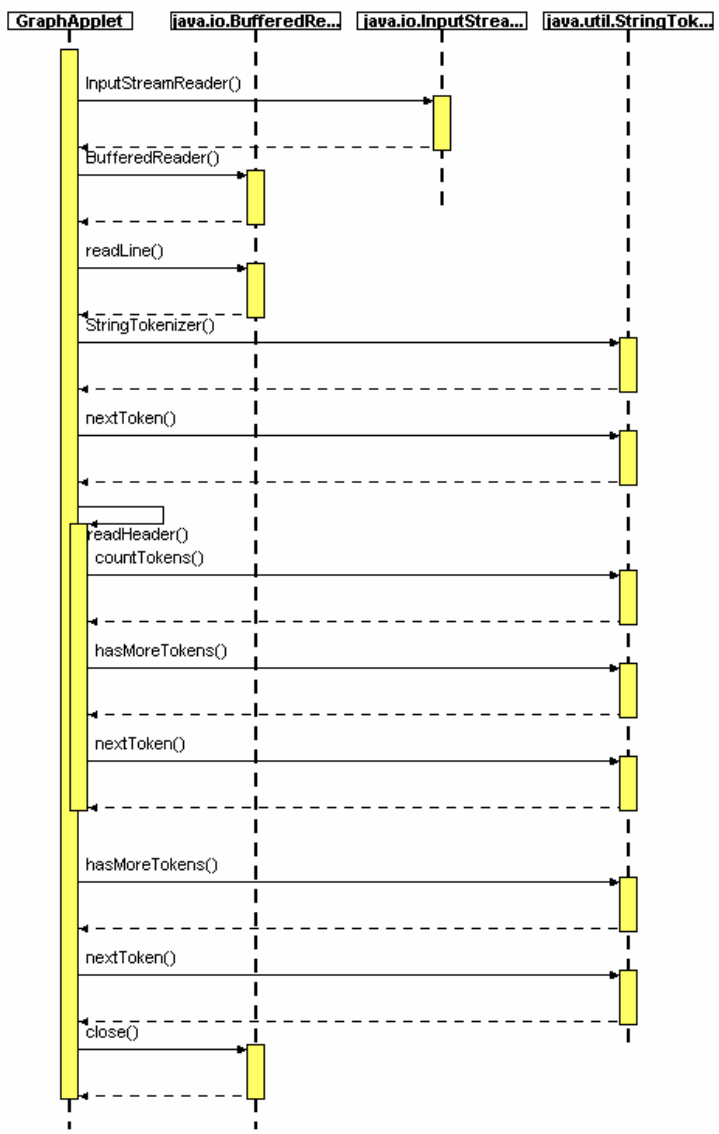


figura 4.22

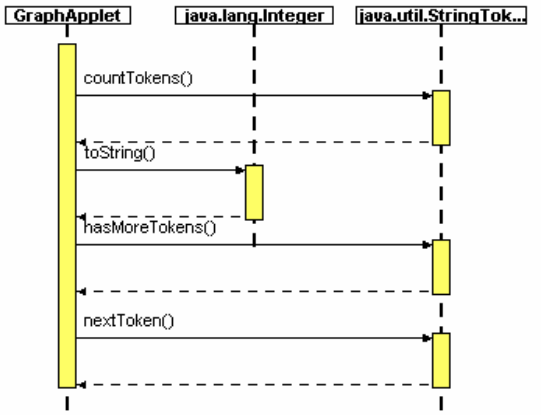


figura 4.23

In figura 4.24 è illustrato il diagramma temporale del metodo costruttore della classe MyCeckbox, tramite il quale gli si associa un ascoltatore di evento. Questo oggetto è una sottoclasse della classe principale java.awt.Checkbox ed è stato costruito per aggiungere, appunto, l'ascoltatore di evento alle funzionalità di base offerte dal componente Checkbox.

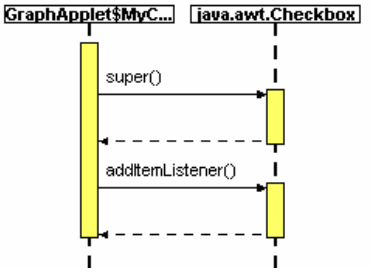


figura 4.24

Concluse le operazioni del metodo init2(), viene richiamata la procedura Reset(), il cui diagramma temporale è illustrato in figura 4.25.

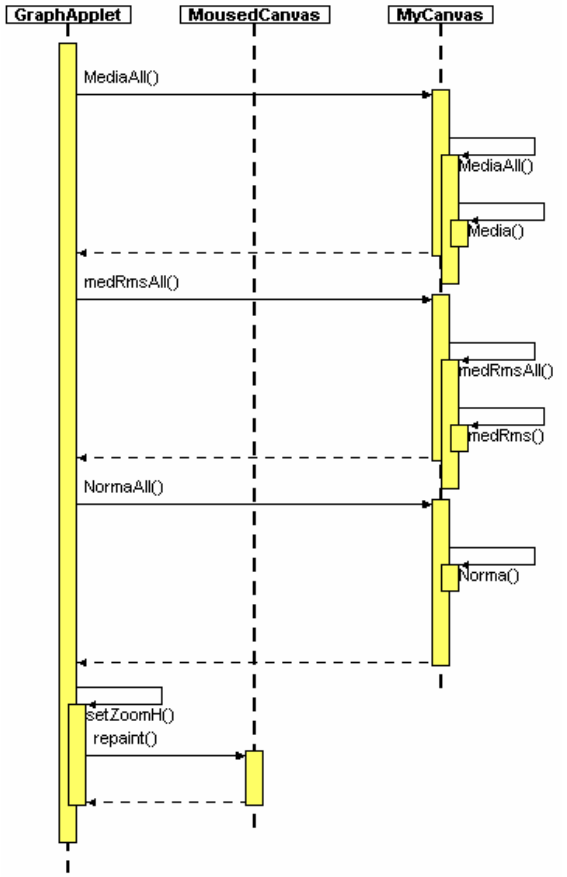


figura 4.25

Tale procedura, avviata all'inizio del ciclo di vita dell'applet è necessaria per far sì che ogni sismogramma sia visualizzato correttamente. Infatti, misurando media statistica ed errore quadratico medio di ogni sismogramma, si fa in modo da eliminare la componente continua e, quindi, centrare la forma d'onda sul

pannello di visualizzazione ad esso associato. Il metodo Norma(), poi, è usato per determinare il valore più grande tra

$$|MAXi| \text{ e } |MINi|$$

dove |MAXi| e |MINi| sono, rispettivamente, il massimo e il minimo valore d'ampiezza del sismogramma i-esimo, in modulo. Determinare queste due grandezze è utile per la successiva operazione di adattamento a fondo scala dei sismogrammi. Per fare ciò, infatti, si normalizzano i campioni per il più grande tra |MAX| |MIN|, in modo tale che i segnali varino nell'intervallo (-1,1). Lo scopo di una tale operazione è quello di poter visualizzare a fondo scala i sismogrammi. L'idea parte dal presupposto di schematizzare l'intera area di visualizzazione dell'applet in una sorta di array (Nx1), dove la riga i-esima rappresenterebbe il pannellino di visualizzazione del sismogramma i-esimo. Quindi il fondo scala sarebbe dato dalla semplice equazione:

$$FS=0.5*(NmPV/NT)$$

con NmPV il numero massimo di pixel a disposizione sull'asse verticale dell'area di visualizzazione dell'applet e NT il numero di tracce da visualizzare. Per cui moltiplicando tutti i campioni, precedentemente normalizzati, per FS, si ottiene che l'ampiezza di un sismogramma vari al più nell'intervallo (-FS,FS).

Infine, nelle figure 4.26 e 4.27 sono illustrate le operazioni che vengono compiute nel tempo, quando un utente agisce sui bottoni "DC", "Norma", mentre per quanto riguarda le funzionalità del bottone "Reset", queste sono già state illustrate in figura 4.25.

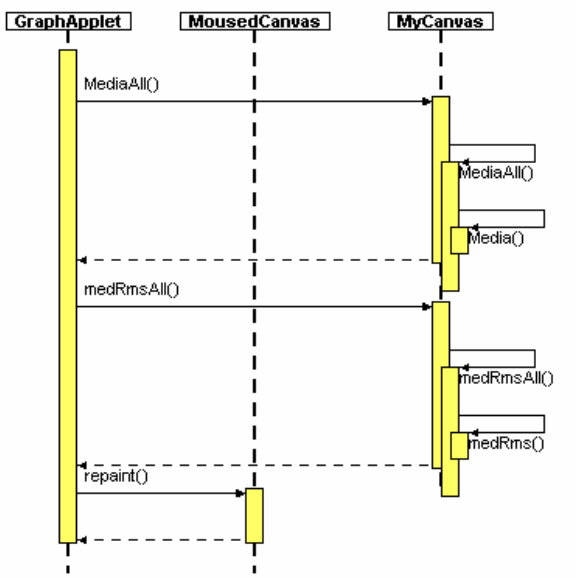


figura 4.26

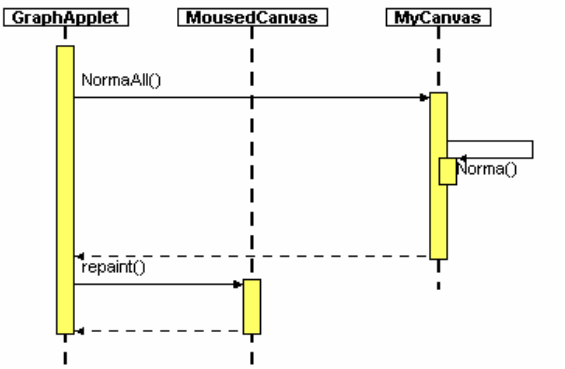


figura 4.27

4.4 MyCanvas.java

In questo paragrafo vengono descritte le funzionalità dell'oggetto MyCanvas, necessarie per "disegnare" a video i sismogrammi a partire dai loro campioni.

In particolare, in figura 4.28 viene illustrato il metodo "paint()", per disegnare a video i sismogrammi.

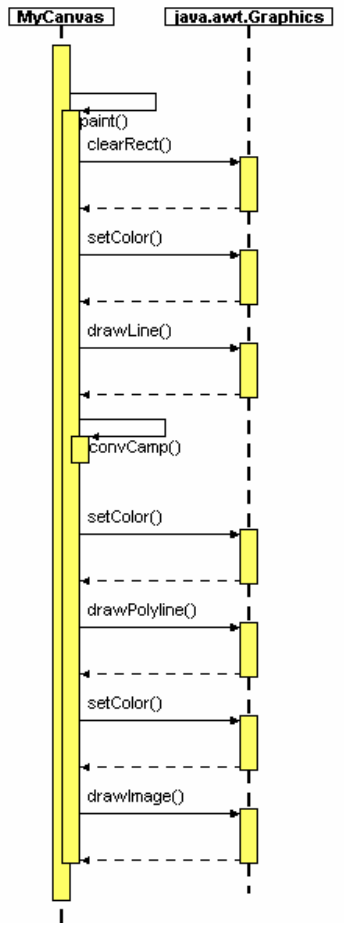


figura 4.28

Si può notare l'uso del metodo "convCamp()". Esso viene utilizzato per effettuare l'adattamento a fondo scala dei sismogrammi e la conversione in pixel dei campioni di ogni sismogramma. A tale scopo viene prima determinato, con il metodo Norma(), il più grande valore tra |MAXi| e |MINi|, dell'i-esimo sismogramma, e creato un array (Nx1), con N il numero dei sismogrammi da visualizzare, indicato con "ZoomSingle", contenente l'inverso di tale valore . Fatto ciò, tutti i campioni dell'i-esimo sismogramma vengono moltiplicati per l'i-esimo elemento dell'array ZoomSingle e moltiplicati per FS.

4.5 MousedCanvas.java

In figura 4.29 è illustrato il metodo costruttore della classe MousedCanvas, necessaria per estendere le funzionalità di MyCanvas. E' grazie a questa classe che si possono gestire gli eventi di mouse sull'area in cui sono stati disegnati i sismogrammi.

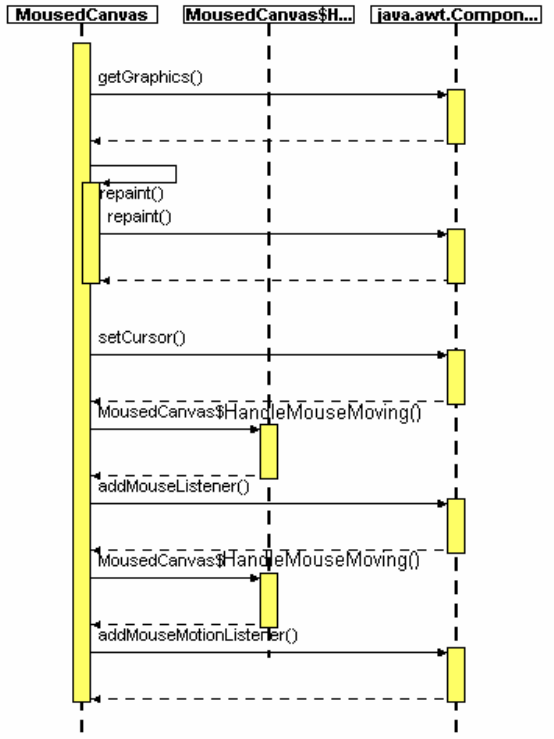


figura 4.29

In particolare, per far ciò viene creata una ulteriore classe, HandleMouseMoving, che estende la classe MouseAdapter e implementa l'interfaccia MouseMotionListener.

In questa classe vengono gestiti:

- i movimenti del mouse, evidenziandoli con un cursore grafico;
- l'ingresso e l'uscita del cursore nelle aree dedicate ai sismogrammi;
- la pressione e il rilascio dei pulsanti del mouse nelle aree suddette;

La figura 4.30 illustra il metodo "mouseMoved()", per la gestione del movimento del mouse.

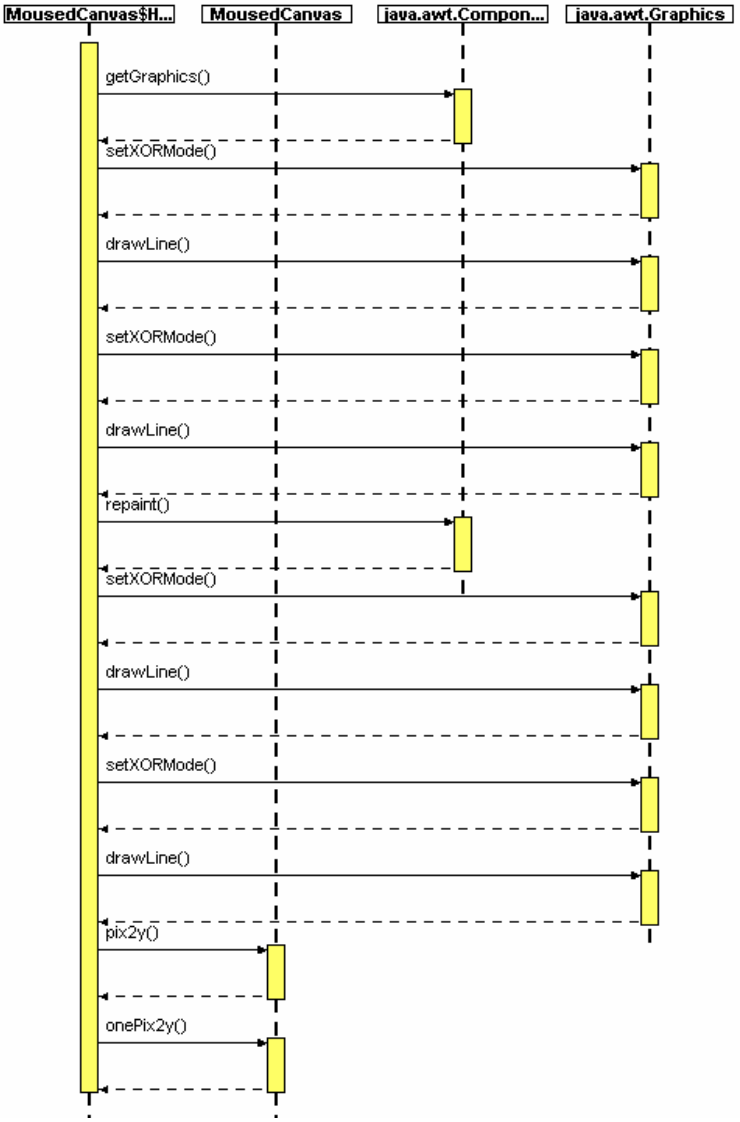


figura 4.30

In particolare vengono gestite due azioni, al muoversi del mouse su un sismogramma:

- il cursore cambia colore, quando si sovrappone alla forma d'onda;
- viene rilevata l'ampiezza del sismogramma nella posizione corrente del mouse.

La prima operazione la si ottiene grazie al metodo "setXORMode()" della classe di java.awt.Graphics, mentre la seconda convertendo il pixel, rappresentativo del campione del sismogramma, puntato dal mouse grazie al metodo "pix2y()", per poter visualizzare tale informazione in un'apposita area di testo dell'applet.

In figura 4.31 è rappresentato il diagramma temporale del metodo "mousePressed()".

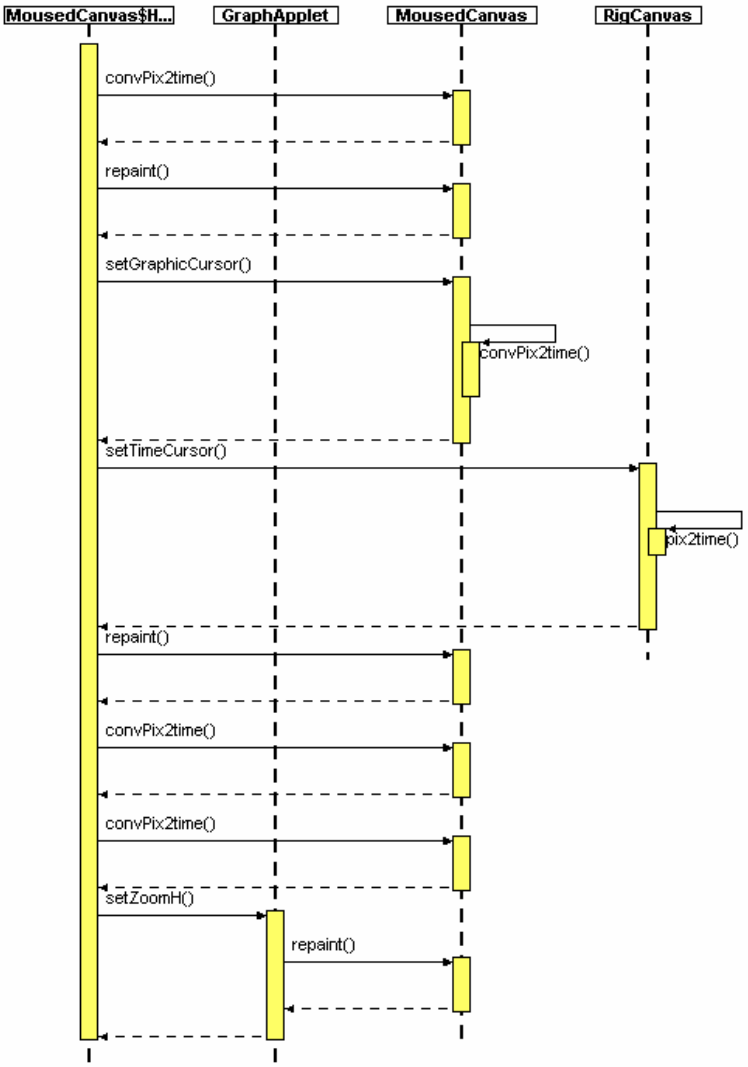


figura 4.31

Tale metodo gestisce gli eventi di pressione dei pulsanti del mouse, quando o si sta effettuando un operazione di zoom o di picking, abilitando una delle due operazioni con apposite checkbox dell'applet. Nel primo caso si seleziona, con due click del mouse l'area da ingrandire e, tramite il metodo "setZoomH()", si ridisegna a fondo scala quella parte del sismogramma selezionata.

Grazie al metodo "setTimeCursor()" si setta un "cursore temporale", tramite il quale è possibile visualizzare l'istante di tempo relativo alla posizione del mouse sul sismogramma.

In figura 4.32 e 4.33 sono illustrati i metodi "mouseEnter" e "mouseExited", rispettivamente.

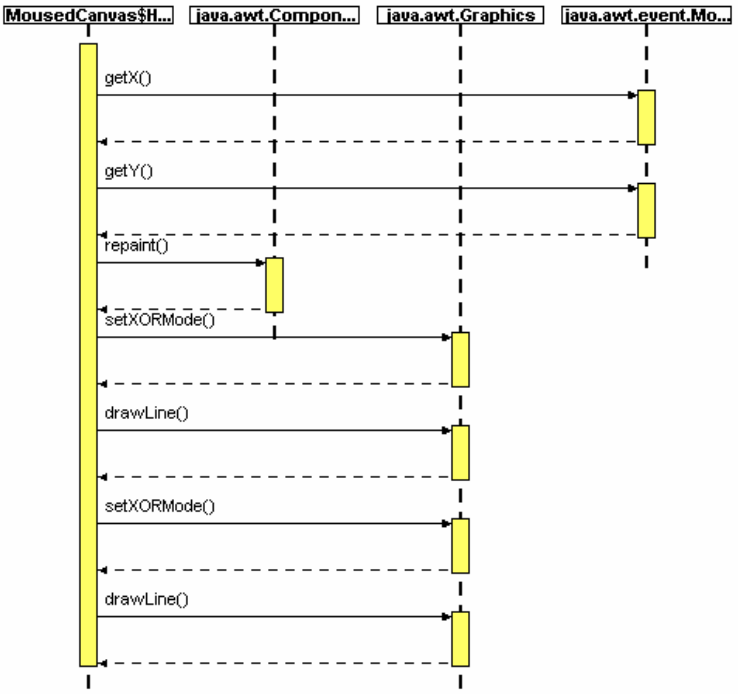


figura 4.32

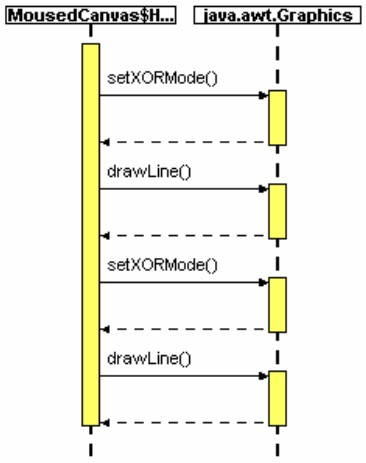


figura 4.33

La funzione di tali metodi è quella di notificare quando si presentano gli eventi di ingresso o uscita del mouse, rispettivamente, in una particolare area dell'applet dei sismogrammi.
Il metodo "paint()", in figura 4.34 (una volta richiamato il metodo paint() della sua superclasse, MyCanvas()), disegna a video un cursore grafico che segue i movimenti del mouse.

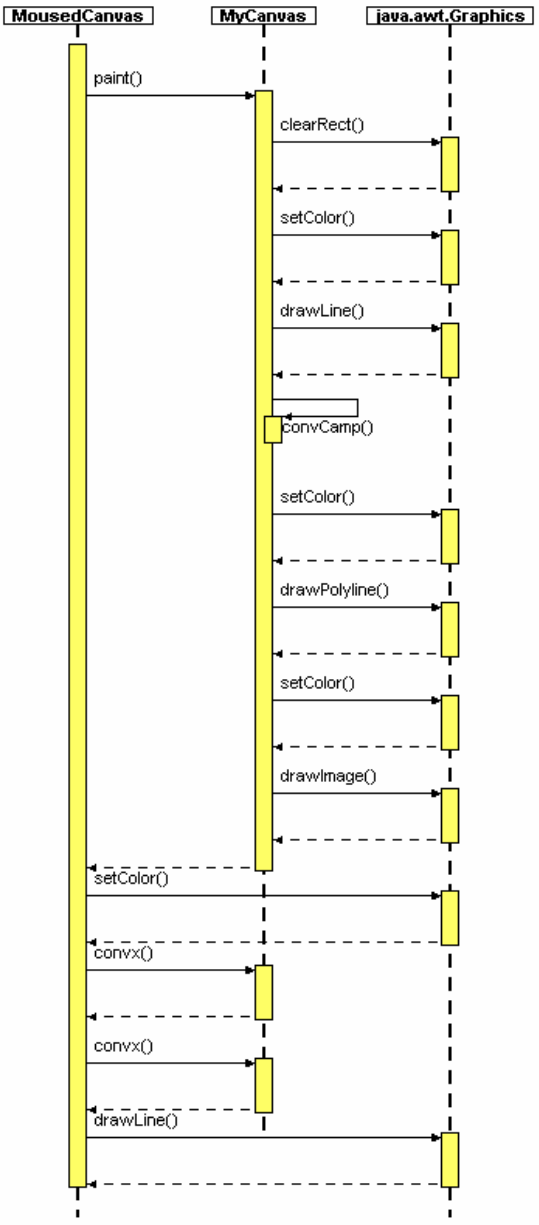


figura 4.34

4.6 RigCanvas.java

Come accennato nel paragrafo precedente, nella gestione dei movimenti del mouse si tiene conto delle informazioni temporali del sismogramma. Infatti, grazie alla classe RigCanvas si aggiunge all'applet un "righello" su cui vengono mostrate le informazioni temporali relative alla posizione corrente del mouse sul sismogramma. In questo modo si può leggere l'istante di tempo relativo al campione, del sismogramma, puntato dal mouse, tradotto in orario GMT.
Inoltre, cliccando una volta con il mouse in un punto e spostando poi il cursore, è possibile visualizzare anche la durata, in secondi, della porzione di sismogramma compresa tra il primo click e la posizione corrente del cursore. Ciò permette di misurare la durata di un evento, nel caso si stia effettuando un'operazione di picking, o di misurare la durata della porzione di sismogramma che si vuole ingrandire, nel caso si stia effettuando un'operazione di zoom.

4.7 MyScrollbar.java

MyScrollbar è una classe per la gestione degli scrollbar verticale ed orizzontale dell'applet grafica. Infatti, tramite questa classe è possibile stabilire la quantità di pixel da "shiftare" in verticale o in orizzontale, quando l'aspetto dei sismogrammi cambia in seguito ad un operazione di zoom. In figura 4.35 è illustrato il metodo costruttore della classe MyScrollbar.
In tale figura si possono notare gli oggetti "HandleScrollingx" e "HandleScrollingy", necessari per la gestione degli eventi d'utente sugli scrollbar orizzontale e verticale, rispettivamente.
In pratica, se si sposta orizzontalmente lo scroller, si scorre lungo un sismogramma con un passo dato dal numero di pixel settato nella classe "HandleScrollingx", mentre se si sposta verticalmente lo scroller, si scorre tra i diversi sismogrammi dell'applet.

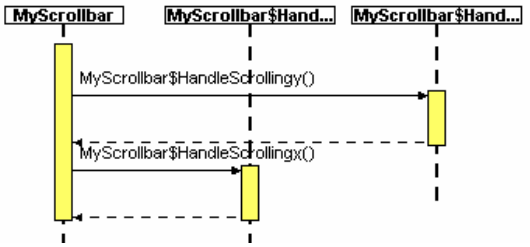


figura 4.35

4.8 MyPanel.java

In figura 4.36 viene illustrato il diagramma temporale del metodo costruttore della classe MyPanel.

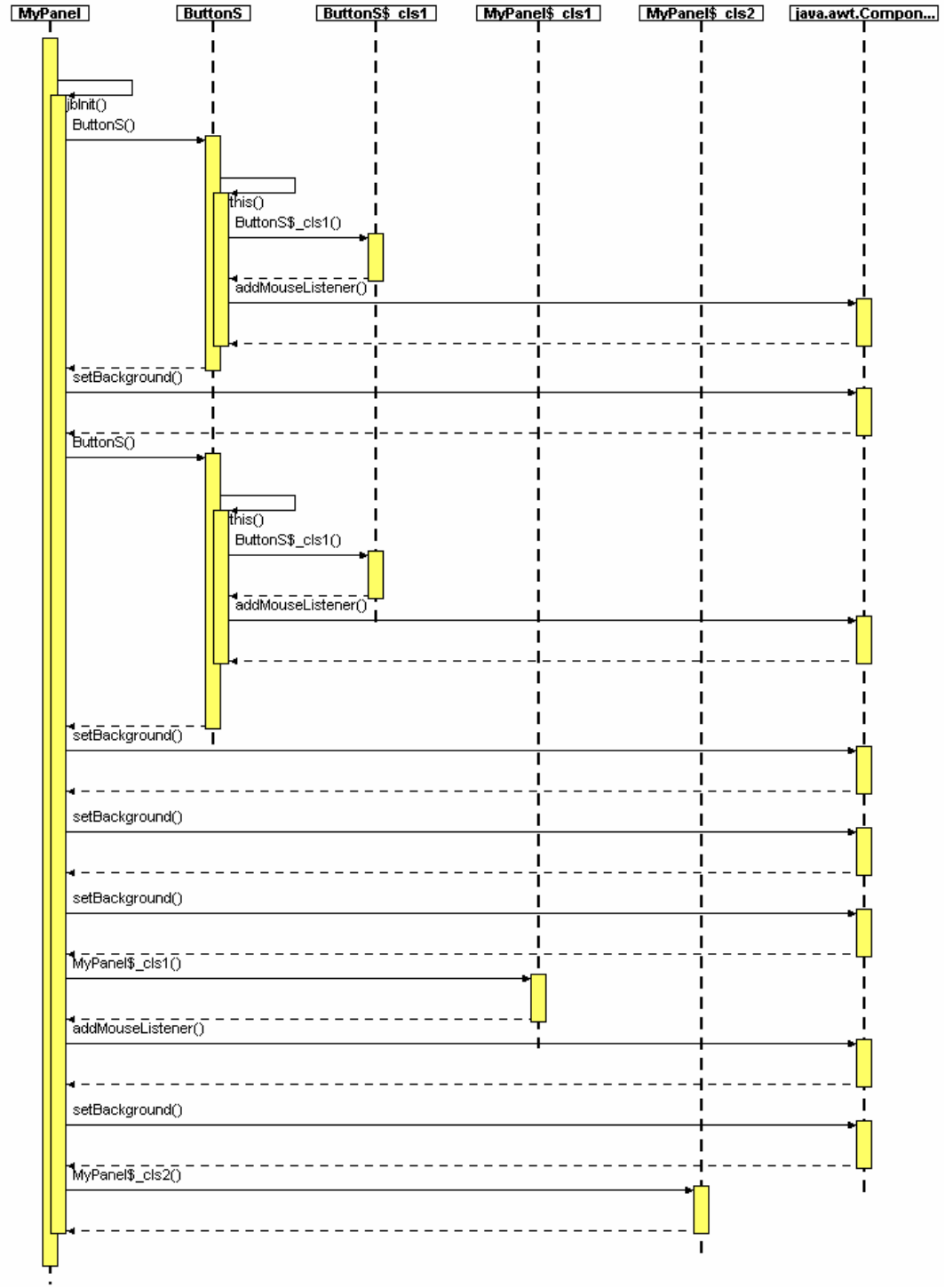


figura 4.36

Questa classe personalizza la classe predefinita di Java, java.awt.Panel, estendendone le funzionalità.
Le istanze di MyPanel, graficamente, rappresentano dei pannelli alla sinistra dall'area grafica dedicata al singolo sismogramma, sui quali vengono visualizzate le informazioni sulla stazione, sul canale e sulla rete, relative al sismogramma ad esse associato. Inoltre su ogni pannello, tanti quanti sono i sismogrammi da visualizzare, vengono aggiunti tre bottoni, istanze della classe ButtonS, i cui dettagli saranno discussi nel prossimo paragrafo. Grazie a questi tre bottoni, posizionati su ogni pannello, è possibile effettuare delle particolari operazioni sismogramma per sismogramma. In particolare è possibile rimuovere la componente continua, amplificare o attenuare solo particolari sismogrammi, anziché effettuare le stesse operazioni contemporaneamente su tutte le tracce, cosa che, invece, accade agendo sui bottoni introdotti in GraphApplet.
In figura 4.37 è illustrato il diagramma temporale del metodo init().

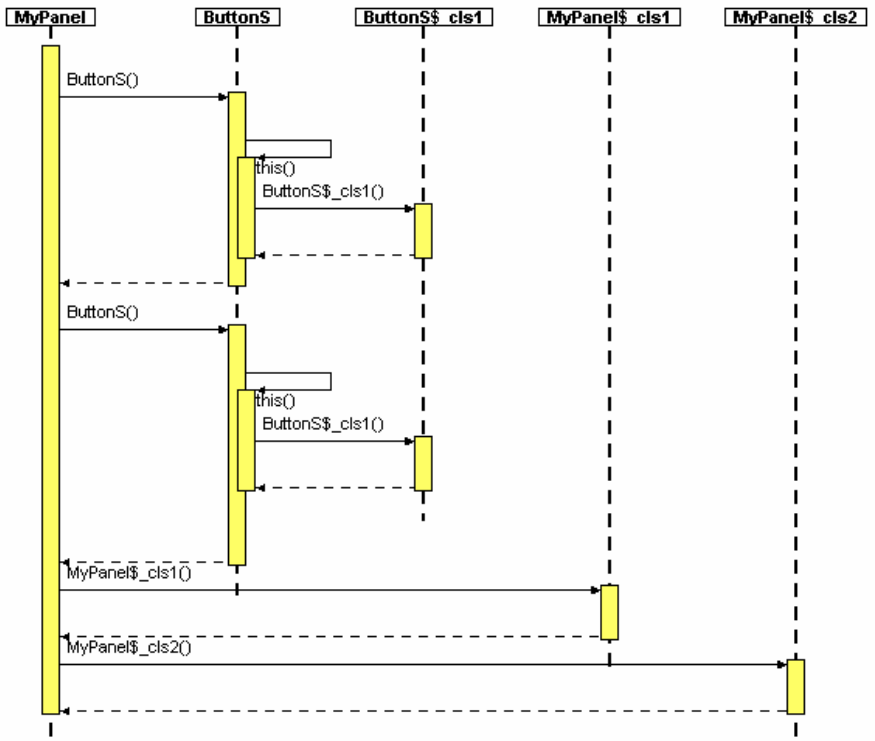


figura 4.37

Dalla figura si può notare come venga istanziata due volte la classe ButtonS: la prima rappresenta il bottone di amplificazione "+", la seconda di attenuazione "-".

La classe _cls1 (MyPanel) estende la classe java.awt.event.MouseAdapter ed è creata per la gestione degli eventi del mouse sui componenti. In particolare grazie ad essa viene rilevato quando il puntatore del mouse entra in una determinata area di un sismogramma e passato all'applet GraphApplet la terna Station-Channel-Network, affinché tale applet possa notificare all'utente l'avvenuto ingresso del mouse in quell'area. Tale notifica avviene settando tre campi di testo, etichettati rispettivamente con Station, Channel, Network.
In figura 4.38 è illustrato il diagramma temporale della classe _cls1.

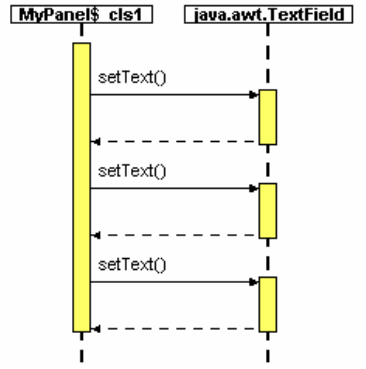


figura 4.38

In figura 4.39 sono illustrati i dettagli della classe _cls2, per gestire la azioni che scaturiscono dalla pressione del bottone "DC", su un MyPanel.

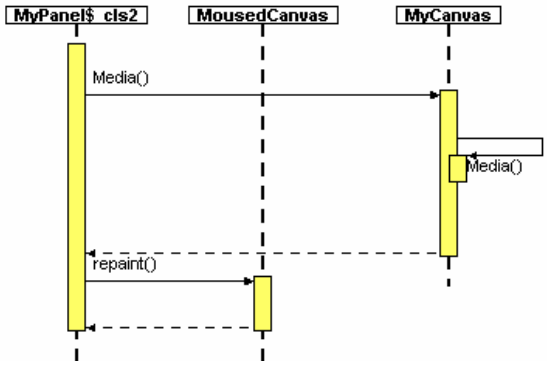


figura 4.39

Come si può notare, agendo sul bottone "DC" si avvia la procedura "Media()" di MyCanvas, con la quale si rimuove la continua al sismogramma.

4.9 ButtonS.java

In figura 4.40 viene illustrato il metodo costruttore della classe ButtonS.

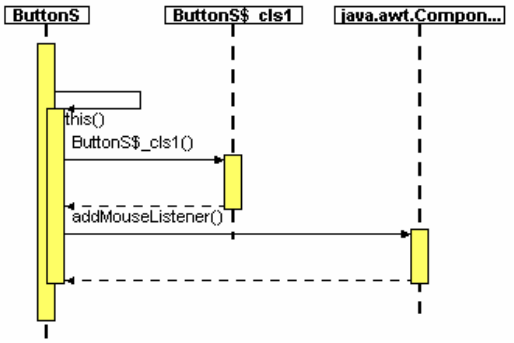


figura 4.40

Da questa si nota che alla classe ButtonS viene aggiunto l'ascoltatore di eventi, implementato dalla classe _cls1. Infatti _cls1 estende la classe java.awt.event.MouseAdapter ed implementa i metodi "mousePressed()" e "mouseReleased()", illustrati in figura 4.41 e 4.42.

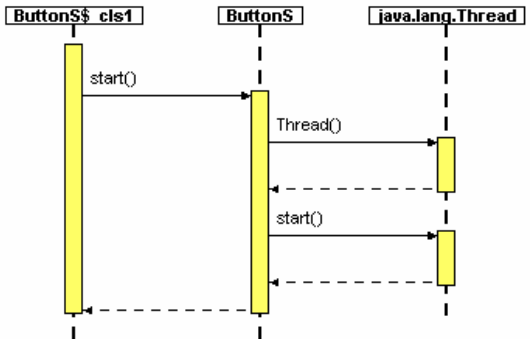


figura 4.41

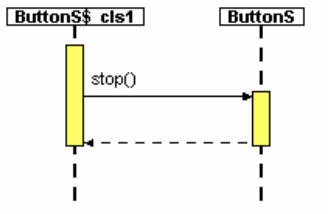


figura 4.42

In questo modo, quando si preme un bottone, istanza di ButtonS, si avvia una procedura, start(), della classe java.lang.Thread, grazie alla quale è possibile

svolgere delle azioni, mentre il resto del programma effettua altre operazioni. Questa procedura manda automaticamente in esecuzione il metodo run(), grazie al quale è possibile, ad esempio, "zoommare" continuamente un sismogramma, finché questo bottone resta premuto.

Capitolo 5

CONCLUSIONI

DiVARES è la realizzazione di uno strumento di accesso remoto ai dati sismici e di analisi *off-line*, che consente di effettuare l'analisi da remoto degli eventi sismici e di correggere eventuali errori nelle localizzazioni automatiche.

Il sistema, sviluppato con tecnologie Web-based, è in grado di interfacciarsi ad Earthworm, in uso presso il Centro di Monitoraggio dell'Osservatorio Vesuviano, in modo da estrarre le forme d'onda degli eventi sismici e presentarle all'analista sismologo. In questo modo l'utente può effettuare una serie di operazioni di analisi e di localizzazione degli eventi, con l'opportunità di apportare le dovute correzioni alle mancate o errate localizzazioni automatiche, sulla base della propria esperienza. DiVARES, dunque, è volto a coniugare in maniera ottimale l'esperienza dell'operatore con l'efficienza delle infrastrutture informatiche.

In tal senso, la scelta del Web come mezzo di comunicazione, tra i servizi messi a disposizione dalla macchina remota e l'utente finale offre vantaggi, quali capillarità, indipendenza dalla piattaforma client, economia.

DiVARES consente di visualizzare tutti i segnali provenienti dalle stazioni, di effettuare alcune analisi preliminari ed è predisposto per effettuare nuove localizzazioni. L'idea si basa sulla possibilità di inviare i parametri necessari per la localizzazione ad opportuni moduli server, che hanno la funzione di interfaccia verso i sistemi di analisi automatica, in modo da attivare le procedure di localizzazione. I risultati di tali localizzazioni possono essere controllati con gli strumenti per il rendering, previsti dai preesistenti sistemi (WBSM).

L'architettura del sistema in oggetto si compone di più moduli software sviluppati in Java. Alcuni di questi sono moduli residenti sulla macchina server in grado di svolgere operazioni di interfacciamento verso i sistemi automatici di analisi, estrazione dei dati "grezzi" delle informazioni parametriche ed inizializzazione ed avvio dei moduli lato *client*.

I moduli lato *client* danno l'opportunità di visualizzare, da remoto, i sismogrammi e inviare i parametri necessari alla localizzazione ad appositi moduli lato server, oltre che rilevare tutti i parametri associati all'evento. Essi, dunque, svolgono le operazioni di interazione grafica con l'utente, che non richiedono un accesso diretto ai dati, mentre è affidato ai moduli server

il reperimento delle informazioni che saranno messe a disposizione dell'utente, o sotto forma di file contenente i campioni dei sismogrammi, o in maniera grafica, in modo da permettere all'utente la visualizzazione e l'analisi. Per le caratteristiche di modularità, DiVARES è facilmente integrabile anche in altri sistemi diversi da quelli in uso presso l'Osservatorio Vesuviano. Infatti le modalità di interfacciamento del dispositivo sono regolate da appositi moduli software, che, opportunamente modificati, consentono l'accesso ad una qualunque altra fonte di dati.

E' prevista l'ottimizzazione della gestione dei file creati durante le operazioni di *downloading* o *rendering*, adottando un sistema di compressione dei file e creare un archivio per l'accesso *off-line*. Infine, si potrebbe prevedere una modalità protetta di accesso al sistema, in modo da riservare il servizio di downloading ed analisi delle forme d'onda ai soli utenti.

Attualmente l'applicazione è disponibile, in fase di test, all'indirizzo provvisorio http://193.206.115.249:8585/divares.html.

BIBLIOGRAFIA

[1] - Marty Hall, ***Core Servlets and JavaServer Pages***, Ed. Prantice Hall and Sun Microsystems

[2] - Laura Lemay, Rogers Cadenhead, ***Java2 guida completa***, Ed. APOGEO

[3] - http://jakarta.apache.org

[4] - http://java.sun.com

[5] - http://www.html.it

[6] - http://www.moreservlets.com/Using_Tomcat-4.html

[7] - http://www.seismosoc.org/publications/IASPEI_Software.html

[8] – http://gldbrick.cr.usgs.gov/ovr/wave_serverV_ovr.html

[9] – Giudicepietro et al., 2000 - Il SIstema Sismometrico Modulare Integrato (SISMI) – http://www.ov.ingv.it/italiano/pubblicazioni/openfile/ofr_06_00.htm