Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Progettazione e implementazione di nuove funzionalità per un'applicazione web di visualizzazione di dati sperimentali

Laureando

Marco Cioccarelli

Relatore

Prof. Michele Moro

Correlatore

Ing. Gabriele Manduchi

Anno accademico 2014/2015

Sommario

L'elaborato descrive i risultati dell'attività di tesi svolta presso l'Istituto Gas Ionizzati del CNR di Padova con l'obiettivo di sviluppare nuove funzionalità per l'applicazione di visualizzazione dati WebScope, che è parte del sistema software MDSplus di acquisizione, memorizzazione e gestione di dati sperimentali. Le principali funzionalità implementate sono la nuova e più efficiente gestione dell'accesso ai dati, la visualizzazione di sequenze di frame e la visualizzazione di dati in tempo reale.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Nell'ambito della ricerca scientifica, la visualizzazione dei dati acquisiti durante gli esperimenti o prodotti nelle simulazioni assume una notevole importanza. Uno strumento di visualizzazione efficace semplifica la comprensione e l'analisi dei dati, consente di individuare pattern, effettuare confronti e identificare rapporti di causa-effetto, anche nel caso di volumi di dati molto grandi. Di conseguenza, le applicazioni di visualizzazione sono uno dei principali strumenti software usati dai ricercatori e sono spesso incluse nei sistemi di acquisizione e analisi dei dati, come il sistema MDSplus[1] di cui fa parte l'applicazione oggetto di questa tesi.

MDSplus è il sistema software più usato per la gestione e la memorizzazione dei dati nel campo della ricerca sulla fusione termonucleare controllata, con installazioni in oltre 30 istituti in tutto il mondo. Alla prima applicazione di visualizzazione dati integrata in MDSplus, chiamata DwScope, se ne è in seguito aggiunta una più avanzata, jScope, sviluppata in Java e quindi eseguibile su tutte le principali piattaforme. Una delle caratteristiche introdotte in jScope è la possibilità di accedere ai dati presenti in un server remoto, attraverso un protocollo basato su TCP/IP chiamato MDSip. L'abilitazione di questa funzionalità comporta però alcuni inconvenienti, come la necessità di disattivare la protezione da parte del firewall su alcune porte, cosa che può produrre dei rischi per la sicurezza. È questo uno dei motivi che ha portato alla realizzazione di WebScope, un'applicazione web per la visualizzazione dei dati che, usando il protocollo HTTP, non richiede accorgimenti particolari per consentire l'accesso ai dati. WebScope rende possibile la visualizzazione dei dati presenti su un server MDSplus da qualsiasi dispositivo, compresi quelli touchscreen come smartphone e tablet, senza la necessità di installare un'applicazione dedicata, e rappresenta quindi lo strumento ideale per l'accesso remoto che va ad affiancarsi a jScope, maggiormente indicato per un utilizzo all'interno dei laboratori.

Essendo un'applicazione relativamente recente, WebScope non include ancora tutte le funzionalità richieste. L'obiettivo di questa tesi è l'implementazione di alcune funzionalità di particolare importanza, ovvero la modifica del procedimento con cui viene effettuato l'accesso ai dati, per renderlo più efficiente, la visualizzazione di dati che rappresentano

[1] L'acronimo MDS sta per Model Driven System e il suo significato sarà chiarito nel capitolo 2.

sequenze di frame, l'aggiornamento delle informazioni visualizzate in tempo reale, per consentire il monitoraggio durante il processo di acquisizione, e altre funzionalità minori.

L'attività di tesi è stata svolta presso l'Istituto Gas Ionizzati, che collabora con il Massachusetts Institute of Technology e il Los Alamos National Laboratory nello sviluppo di MDSplus. L'Istituto Gas Ionizzati, situato nell'Area della Ricerca del CNR di Padova, è parte del Consorzio RFX, che comprende CNR, ENEA, Università degli Studi di Padova, Acciaierie Venete S.p.A. e INFN, fondato a Padova nel 1996 per facilitare la collaborazione tra i vari enti e imprese nell'ambito dell'associazione con Euratom, l'organizzazione europea che coordina i programmi di ricerca degli stati membri dell'UE per l'uso pacifico dell'energia nucleare. Il Consorzio si occupa della ricerca scientifica e tecnologica sulla fusione termonucleare controllata come possibile fonte di energia, in particolare approfondendo le conoscenze sui plasmi tramite l'esperimento RFX (Reversed Field eXperiment), contribuendo al progetto internazionale ITER con il design e la realizzazione del prototipo di iniettore di neutri (Neutral Beam Injector, NBI) per il riscaldamento e il controllo del plasma, e sviluppando nuove tecnologie e dispositivi a supporto dell'attività di ricerca.

Il seguito della tesi è organizzato in questo modo: il capitolo 2 presenta le caratteristiche del sistema MDSplus, il capitolo 3 introduce le tecnologie usate nello sviluppo di WebScope, il capitolo 4 descrive la versione iniziale dell'applicazione e il capitolo 5 espone le nuove funzionalità implementate durante l'attività di tesi.

Capitolo 2

MDSplus

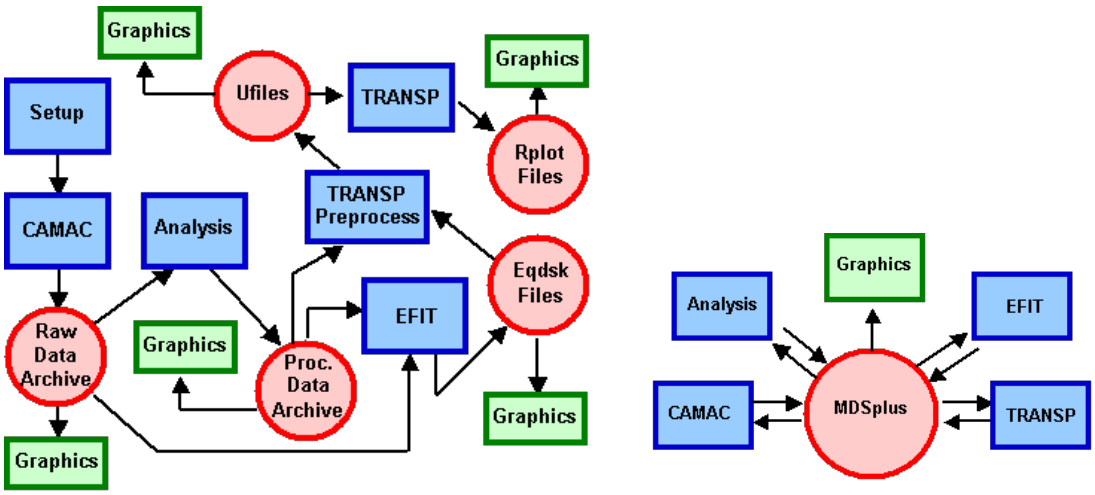
Questo capitolo introduce i concetti generali e le caratteristiche del sistema MDSplus.

2.1 Introduzione

MDSplus è un sistema di acquisizione, memorizzazione e gestione di dati sviluppato dal Massachusetts Institute of Technology, dall'Istituto Gas Ionizzati di Padova e dal Los Alamos National Laboratory. Usato da oltre 30 istituti di ricerca in tutto il mondo, è il sistema di gestione di dati più diffuso nell'ambito della ricerca sulla fusione termonucleare controllata.

Il sistema MDSplus permette di creare e mantenere dataset completi, coerenti e autodescrittivi, consentendo di memorizzare nella stessa struttura tutti i dati relativi ad un esperimento o una simulazione, che siano informazioni relative al setup e alla calibrazione, dati grezzi e processati o descrizioni e *scheduling* di task. Mette inoltre a disposizione un'unica interfaccia per l'accesso ai dati, facilitando la loro condivisione e l'interoperabilità tra le diverse applicazioni che ne fanno uso e supportando numerose piattaforme e linguaggi di programmazione. Rispetto ad altri sistemi di memorizzazione e gestione di dati, MDSplus si distingue per la struttura gerarchica in cui vengono mantenuti i dati, che consente un'interazione con l'utente basata su riferimenti logici anziché fisici per l'accesso ai dati, per l'impiego di metadati, la disponibilità di molti tipi di dato semplici e composti, l'utilizzo di espressioni e la scalabilità fino a dataset molto grandi e complessi.

Le caratteristiche di MDSplus permettono l'introduzione di un nuovo paradigma per l'analisi dei dati, che si contrappone all'approccio tradizionale in cui le varie fasi di elaborazione sono contraddistinte ciascuna da differenti tipi di file, interfacce di accesso e routine di visualizzazione, e in cui di conseguenza risulta difficile riutilizzare applicazioni per fasi diverse e confrontare i risultati ottenuti da diversi strumenti di analisi. Due schemi che mettono a confronto i due approcci nel caso di un particolare tipo di analisi (TRANSP) sono mostrati in figura 2.1, dove le fasi dell'elaborazione sono rappresentate in blu e i diversi tipi di file in rosso: è evidente la semplificazione che l'utilizzo di MDSplus rende possibile.



(a) (b)

Figura 2.1: Confronto tra l'approccio tradizionale e quello permesso da MDSplus per l'analisi. Fonte: [4]

MDSplus favorisce lo sviluppo di applicazioni *data-driven*, in cui i dati stessi controllano il flusso di esecuzione di un programma, consentendo alle applicazioni di essere estremamente flessibili e contemporaneamente limitando la necessità di modificarne il codice. Ad esempio è possibile memorizzare come dati MDSplus i nomi e i parametri che definiscono il comportamento delle funzioni di analisi dei dati, oppure implementare delle rappresentazioni dei dispositivi di acquisizione che mantengono informazioni sui parametri di setup, i riferimenti alle procedure di inizializzazione e memorizzazione dei dati e lo *scheduling* delle operazioni da eseguire.

Il processo di acquisizione e analisi dei dati si basa sul concetto di modello sperimentale, da cui deriva l'acronimo MDS, Model Driven System. Un modello definisce le informazioni per il setup e la struttura di memorizzazione usata per un esperimento, e viene usato come *template* per ciascuno *shot*, che costituisce l'unità di memorizzazione in cui sono effettivamente contenuti i dati prodotti da una singola esecuzione dell'esperimento. Il ciclo di acquisizione viene gestito da un *dispatcher*, che invia ad appositi processi di servizio le azioni da eseguire, le cui descrizioni sono anch'esse contenute nella struttura dati. In generale un ciclo di acquisizione è costituito da tre fasi:

- *initialize*: il file contenente i dati per il nuovo *shot* viene creato copiando quello contenente il modello, il *dispatcher* individua le azioni da eseguire e le invia ai processi di servizio.
- *pulse*: viene avviato il sistema di temporizzazione che gestisce l'hardware per l'esecuzione dell'esperimento e l'acquisizione dei dati.
- *store*: il *dispatcher* invia tutte le azioni relative alla memorizzazione e all'analisi dei dati ottenuti.

2.2 Memorizzazione dei dati

2.2.1 La struttura dati *tree*

In MDSplus i dati sono mantenuti in una struttura ad albero definita dall'utente, chiamata *tree*, che presenta due vantaggi principali: l'organizzazione gerarchica permette di conservare e rendere esplicite le relazioni logiche tra i diversi elementi del dataset, facilitando la ricerca dei dati di interesse e la loro interpretazione, e le applicazioni che accedono ai dati possono ricavare informazioni dalla loro posizione nella gerarchia, in modo che, ad esempio, quando nuovi dati vengono inseriti essi possono essere trattati come quelli presenti allo stesso livello.

Ogni *tree* è composto da un insieme di nodi, caratterizzati ciascuno da un nome, un identificativo numerico che lo individua univocamente all'interno del *tree*, e un tipo di utilizzo, che denota il tipo di dato che il nodo contiene. Esempi di tipi di utilizzo sono *numeric* per valori e array numerici, *text* per stringhe di testo, *signal* per i segnali e *device* per le rappresentazioni dei dispositivi di acquisizione e analisi. Due tipi speciali di nodi, che, al contrario degli altri, non contengono dati, sono *structure*, usato per raggruppare un insieme di nodi in una "ramificazione", e *subtree*, ossia il nodo radice di un sottoalbero. Oltre a contenere i dati veri e propri, ad un nodo possono essere associati dei metadati, che permettono alla struttura dati di essere autodescrittiva. Essi includono le lunghezze degli array, le dimensioni dei segnali (vedi sezione 2.3), le unità di misura dei valori e la data in cui sono stati inseriti i contenuti del nodo.

Ad ogni *shot* in MDSplus è associato uno *shot tree* creato a partire da un *model tree*, che corrisponde al modello sperimentale e i cui nodi, al contrario di quelli degli *shot tree*, non contengono dati. Ciascun *tree* è memorizzato su disco in tre file, che hanno lo stesso nome, del tipo `nome_del_tree_model` per il *model tree* e `nome_del_tree_`N per gli *shot tree*, dove N è il numero dello *shot*, e tre diverse estensioni, `.tree`, `.characteristics` e `.datafile`: il primo contiene le informazioni sulla struttura del *tree*, ossia i nomi dei nodi, le loro posizioni e i loro tipi di utilizzo, il secondo contiene i metadati associati ai nodi, e il terzo i dati veri e propri.

2.2.2 Riferimenti ai nodi

Per accedere ai dati di un nodo nel *tree* è necessario specificare un riferimento ad esso, cosa che può essere fatta in diversi modi. Innanzitutto, ad un nodo possono essere associati uno o più *tag*, ossia dei metadati testuali che devono essere unici all'interno del *tree* e che permettono di riferirsi ad un nodo usando un nome corto, rispetto agli altri tipi di riferimento che, a seconda della complessità della struttura, possono diventare anche molto lunghi. Il tag `TOP` è riservato e in ogni *tree* indica il nodo radice.

Un secondo tipo di riferimento è quello assoluto, che inizia con una barra rovesciata seguita dal tag di un nodo, preceduto opzionalmente da una componente che specifica il nome di un *tree*, del tipo `nome_del_tree::`, e seguito da un elenco di nomi di nodi, in modo simile all'elenco dei nomi delle cartelle nel percorso in un filesystem. Il nome di

ogni nodo nell'elenco è preceduto da un punto se il nodo è di tipo *structure* o *subtree*, altrimenti dai due punti. Esempi di riferimenti assoluti sono:

```
\RFX::TOP.RFX.SETUP.TOROIDAL:CONFIG
\PARAMETERS.CALIB:CALIBRATION
```

dove nel primo è stato specificato anche il nome del *tree* (`RFX`) mentre nel secondo viene omesso e si assume che il tag `PARAMETERS` faccia riferimento ad un nodo nello stesso *tree* del nodo di default. Il concetto di nodo di default è legato all'ultimo tipo di riferimento, quello relativo. Quando si accede per la prima volta ad un *tree*, il nodo di default corrisponde al nodo radice, `TOP`; in seguito è possibile cambiarlo in diversi modi, usando funzioni e comandi messi a disposizione da MDSplus. Un riferimento relativo si differenzia da uno assoluto per il fatto che non inizia con il tag di un nodo e che la risoluzione del percorso comincia dal nodo di default. Ad esempio, se il nodo di default è stato impostato a `\RFX::TOP.RFX.SETUP`, un riferimento di tipo relativo al primo nodo dell'esempio precedente sarà `TOROIDAL:CONFIG`.

2.2.3 Segmenti

Quando dei dati vengono scritti in un nodo che ne contiene già altri, questi ultimi vengono sovrascritti. Esistono però delle situazioni, in particolare le acquisizioni di dati che si prolungano per molto tempo, in cui si vuole poter aggiungere ripetutamente nuovi dati in coda a quelli presenti in un nodo: questo è possibile utilizzando i segmenti, ossia blocchi di dati concatenati in una lista che possono essere aggiunti nel nodo in modo incrementale. Ciascun segmento è caratterizzato da un tempo di inizio, un tempo di fine e una dimensione, ovvero un array dei tempi associati a ciascun dato del segmento. Queste informazioni consentono di rendere più efficienti le letture, perché specificando l'intervallo temporale di interesse è possibile ottenere i dati al suo interno in modo tale che il tempo necessario per l'operazione sia indipendente dalla quantità totale di dati nel nodo; senza usare i segmenti sarebbe invece necessario leggere completamente i dati e scartare poi quelli al di fuori dell'intervallo di interesse.

2.3 Espressioni e tipi di dati

Tutte le interfacce di accesso ai dati in MDSplus sono basate sulla valutazione di espressioni, scritte in un particolare linguaggio chiamato TDI (*Tree Data Interface*). Un semplice esempio di espressione è dato da un riferimento ad un nodo di un *tree*, e la sua valutazione restituisce i dati contenuti nel nodo. Le espressioni possono inoltre essere costituite da valori numerici (`12.567`), espressioni matematiche e logiche contenenti i principali operatori (`10*2-7.3`), array (`[10,20,30]`) e stringhe (`'Hello world'`). Il linguaggio TDI supporta poi l'uso di variabili, le strutture di controllo `if-else`, i cicli `for` e la definizione di funzioni. Numerose funzioni TDI sono incluse in MDSplus e possono essere inserite all'interno di un'espressione, come `SIN(0.6)`, ed è possibile per un utente crearne di nuove; in aggiunta è supportata l'invocazione di funzioni scritte in altri linguaggi di programmazione.

Un'espressione può essere memorizzata come dato in un nodo e offrire così una elevata flessibilità nell'accesso ai dati: ad esempio è possibile applicare un offset di un valore contenuto in un nodo ad un segnale contenuto in un altro nodo usando un'espressione che viene poi memorizzata in un terzo nodo. La valutazione dell'espressione permetterà di ottenere il segnale con l'offset applicato, eliminando la ridondanza che si avrebbe memorizzando un secondo segnale che differisce da quello originale solo per un offset; usare un'espressione ha poi il vantaggio che, se il segnale originale dovesse cambiare, la valutazione dell'espressione restituirebbe in modo trasparente la nuova versione con l'offset applicato.

In MDSplus sono presenti vari tipi di dati, che possono essere categorizzati in semplici e composti. I tipi di dato semplici comprendono i valori interi con e senza segno con una dimensione da 8 a 128 bit, i valori *floating point* a 32 e 64 bit, sia reali che complessi, e le stringhe testuali. Tra i tipi di dato composti, che sono costituiti da strutture di due o più campi, si trovano le rappresentazioni delle azioni inviate dal *dispatcher* nel processo di acquisizione, i riferimenti ai nodi e i loro identificativi numerici, gli intervalli di valori, i segnali e le loro dimensioni. Il tipo di dato `Signal`, di particolare importanza, è formato da tre o più componenti: il valore del segnale, una componente opzionale con i dati grezzi (cioè i dati del segnale così come sono stati acquisiti da un dispositivo) e una o più dimensioni. Il valore del segnale può rappresentare i dati veri e propri oppure essere un'espressione che fa riferimento alla componente con i dati grezzi e la modifica in qualche modo, ad esempio effettuando una conversione da gradi Fahrenheit a gradi Celsius; una dimensione, nel caso di un segnale monodimensionale, rappresenta spesso gli istanti temporali associati a ogni dato, cioè le componenti x dei punti del segnale.

2.4 Interfaccia orientata agli oggetti

L'interfaccia orientata agli oggetti di MDSplus permette di realizzare applicazioni per l'accesso e la modifica di dati nei linguaggi C++, Java e Python. L'interfaccia a oggetti si affianca a quella di tipo procedurale basata sui linguaggi C, Fortran, IDL e Matlab che esisteva in precedenza, e rispetto ad essa consente di sfruttare al meglio l'ampia varietà di tipi di dati presenti in MDSplus. Le classi comprese nell'interfaccia sono `Tree`, `TreeNode`, `Event` e un articolato insieme di classi per la rappresentazione dei diversi tipi di dati, aventi come superclasse comune la classe `Data`.

2.4.1 La classe `Tree`

La classe `Tree` è usata per rappresentare un *tree*, e una sua istanza può essere creata usando come parametri il nome dell'esperimento e il numero di *shot* (con il valore speciale -1 che indica il *model tree* dell'esperimento). Un terzo parametro viene usato per specificare la modalità di accesso al *tree*: la modalità di default, *normal*, permette la lettura e la scrittura dei dati ma non la modifica della struttura del *tree*, cosa che è invece consentita se la modalità è *edit*; con *readonly* si ha accesso in sola lettura, mentre *new* viene usata per indicare che il *tree* specificato non esiste e deve essere creato.

Si può ottenere un oggetto che rappresenta un nodo del *tree* con il metodo `getNode`, usando come parametro una stringa contenente un riferimento al nodo; i riferimenti relativi possono essere usati dopo aver impostato il nodo di default con il metodo `setDefault`, mentre `getDefault` permette di conoscere il nodo di default corrente. Con metodo `getNodesWild` si ottiene una lista dei nodi i cui nomi corrispondono ad un pattern che può contenere dei caratteri *wildcard*. Per aggiungere o eliminare nodi dal *tree* sono disponibili i metodi `addNode` e `deleteNode`, e i cambiamenti effettuati vengono poi scritti su disco con il metodo `write`.

2.4.2 La classe `TreeNode`

I nodi di un *tree* sono rappresentati da istanze della classe `TreeNode`, ottenibili con il metodo `getNode` descritto prima o facendo uso del costruttore della classe, che riceve come parametri il *tree* e l'identificativo numerico univoco del nodo. I dati contenuti nel nodo, rappresentati da un'istanza della classe `Data`, possono essere letti con il metodo `getData` e scritti con `setData`, e il loro tipo di dato MDSplus si può ricavare con `getDtype`.

Appartengono alla classe `TreeNode` i metodi che permettono di creare segmenti per la memorizzazione dei dati all'interno del nodo, cosa che è possibile fare in diversi modi. Il metodo `beginSegment` consente di creare un segmento e inserire i dati al suo interno in un'unica operazione, specificando come parametri i tempi di inizio e fine del segmento, la sua dimensione e l'array dei dati. In molte situazioni i dati non sono però disponibili tutti allo stesso momento, perché vengono acquisiti un po' alla volta: il metodo `putSegment` viene usato in questi casi per inserire i dati in un segmento preesistente ad un offset indicato come parametro. Infine, nel caso in cui i dati acquisiti siano associati a dei timestamp, il metodo `putRow` permette la loro memorizzazione in segmenti in modo semplificato: è sufficiente indicare come parametri ciascun dato e il timestamp associato, insieme alla lunghezza scelta per i segmenti; i dati vengono aggiunti in coda a quelli presenti nel segmento corrente e quando questo si riempie ne viene creato automaticamente un altro.

2.4.3 La classe `Data`

Ogni tipo di dato in MDSplus è rappresentato nell'interfaccia orientata agli oggetti da una classe derivata da `Data`. Come mostrato in figura 2.2, dalla classe `Data` derivano direttamente le quattro classi `Scalar`, `Array`, `Compound` e `Apd`: la classe `Scalar` comprende i tipi di dato scalari, ossia gli interi con e senza segno da 8 a 128 bit, i valori *floating point* a singola e doppia precisione e le stringhe, mentre ciascuna delle sottoclassi di `Array` rappresenta un array del corrispondente tipo di dato scalare. Le classi derivate da `Compound` descrivono i tipi di dato composti, come `Signal`, `Dimension`, `Action` e `Dispatch`. La classe `Apd`, acronimo di *Array of Pointers to Descriptors*, consente di memorizzare array di dati eterogenei, in modo da permettere la creazione di strutture dati di qualunque complessità.

La classe `Data` mette a disposizione, per ciascun tipo di dato scalare o array, un metodo che permette la conversione dalla classe dell'interfaccia a oggetti alla rappresentazione

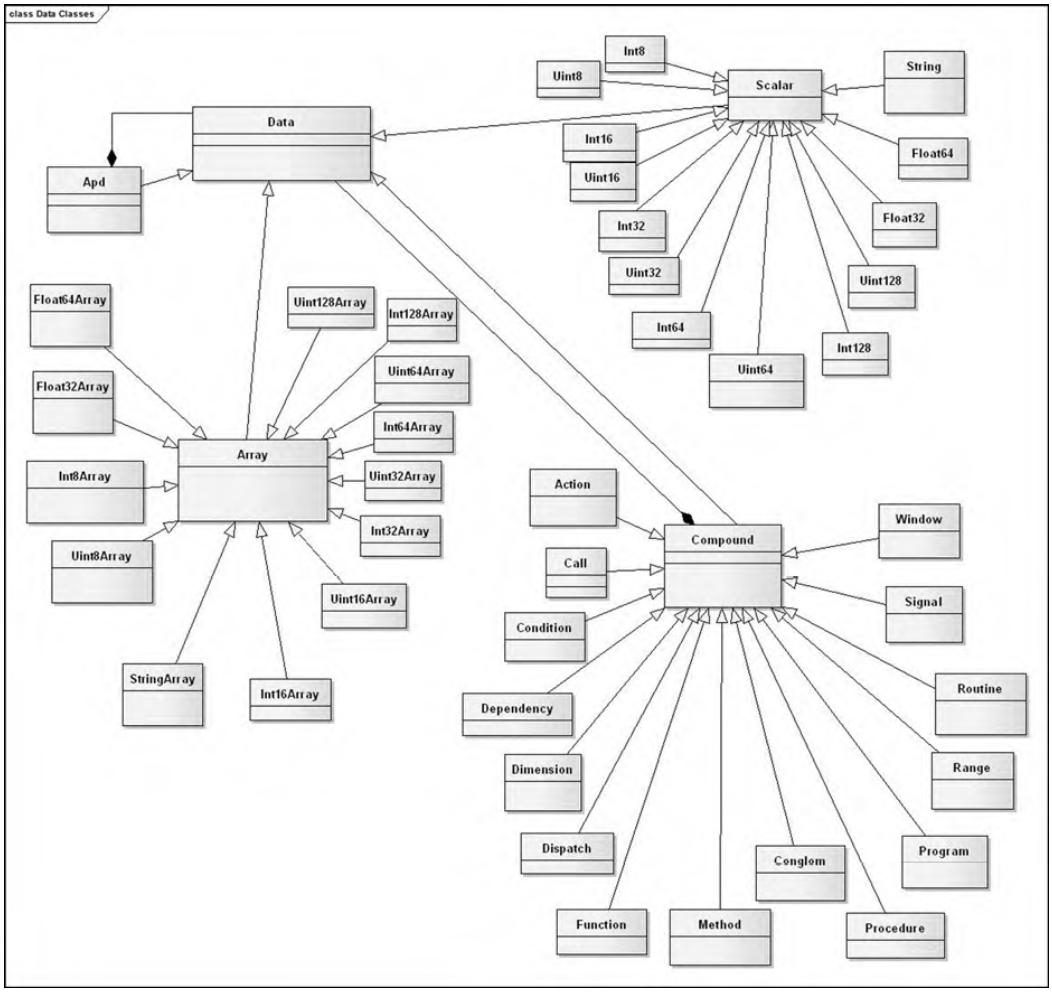


Figura 2.2: Organizzazione delle classi derivate da `Data`. Fonte: [2]

nativa del linguaggio di programmazione usato, in modo da facilitare le successive operazioni da parte di un'applicazione; ad esempio, con il metodo `getDouble` si ottiene il valore contenuto nell'oggetto `Data` come variabile del tipo usato per i numeri *floating point* a 64 bit nel linguaggio usato. Viceversa, si possono creare istanze di una sottoclasse di `Data` usando un valore nativo del linguaggio come parametro per il costruttore.

Le espressioni MDSplus sono rappresentate come strutture ad albero aventi come nodi delle istanze di sottoclassi di `Data`: per esempio, la struttura corrispondente all'espressione `3.5*6` avrà come nodo radice un'istanza della classe `Function` che rappresenta la moltiplicazione e due nodi figli istanze di `Float32` e `Int32`. La creazione di una di queste strutture a partire da una stringa contenente l'espressione è possibile usando il metodo statico `compile` della classe `Data`, mentre il procedimento inverso viene realizzato con il metodo `decompile`. Per la valutazione di un'espressione viene usato il metodo `execute`, che restituisce un oggetto `Data` contenente il risultato della valutazione; ad

esempio, con `Data.execute("\node1")` si ottengono i dati contenuti nel nodo `\node1`, mentre `Data.execute("[1,2,3]*2")` restituisce l'array `[2,4,6]`.

2.4.4 La classe `Event`

In MDSplus è possibile generare eventi asincroni, che possono essere usati per notificare un'applicazione del verificarsi di una certa situazione, come la disponibilità di dati appena acquisiti. Ad ogni evento è associato un nome e, opzionalmente, dei dati. La generazione e la ricezione di eventi avvengono mediante la classe `Event`: per generare un evento si utilizzano i metodi `setevent` e `seteventRaw`, che ricevono come parametri il nome dell'evento e gli eventuali dati, costituiti da un oggetto di tipo `Data` per il metodo `setevent` e da un buffer di byte nel caso di `seteventRaw`. Per gestire la ricezione di un evento è necessario creare una classe derivata da `Event` e definire le azioni da intraprendere all'interno del metodo `run`, che viene invocato ogni volta che si verifica l'evento con il nome indicato come parametro per il costruttore della classe. In seguito alla creazione di un'istanza della classe, essa resterà in attesa dell'evento e, quando questo si verifica, gli eventuali dati potranno essere letti con i metodi `getData` e `getRaw`.

2.5 jTrasverser e jScope

jTrasverser e jScope sono due applicazioni incluse in MDSplus per la gestione dei *tree* e la visualizzazione di dati, scritte in linguaggio Java. jTrasverser permette di navigare la struttura di un tree, visualizzando i nodi presenti, le loro proprietà e i dati in essi contenuti; consente inoltre di modificare il *tree* aggiungendo nuovi nodi ed eliminando o modificando quelli esistenti.

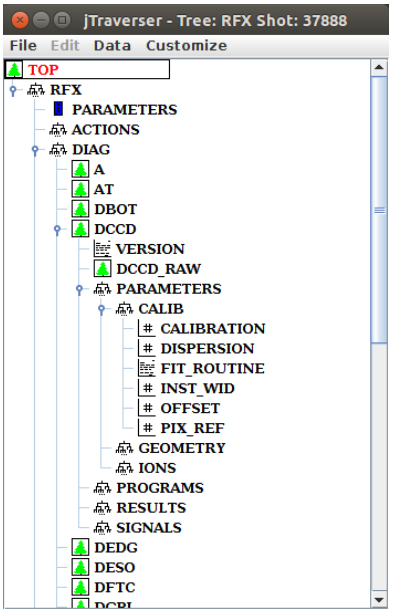


Figura 2.3: Interfaccia di jTrasverser

La figura 2.3 mostra l'interfaccia di jTrasverser con la visualizzazione di parte della struttura di un *tree*. Ad ogni nodo è associata un'icona che ne descrive il tipo di utilizzo.

Usando jScope si possono visualizzare segnali con un'interfaccia suddivisa in un certo numero di pannelli, configurabili dall'utente, ed eseguire operazioni di zoom, pan, crosshair e scaling dell'area visualizzata in ciascun pannello. Le funzionalità di jScope sono molto simili a quelle di WebScope, descritte nel capitolo 4, dato che WebScope è un'applicazione creata prendendo come modello di riferimento jScope.

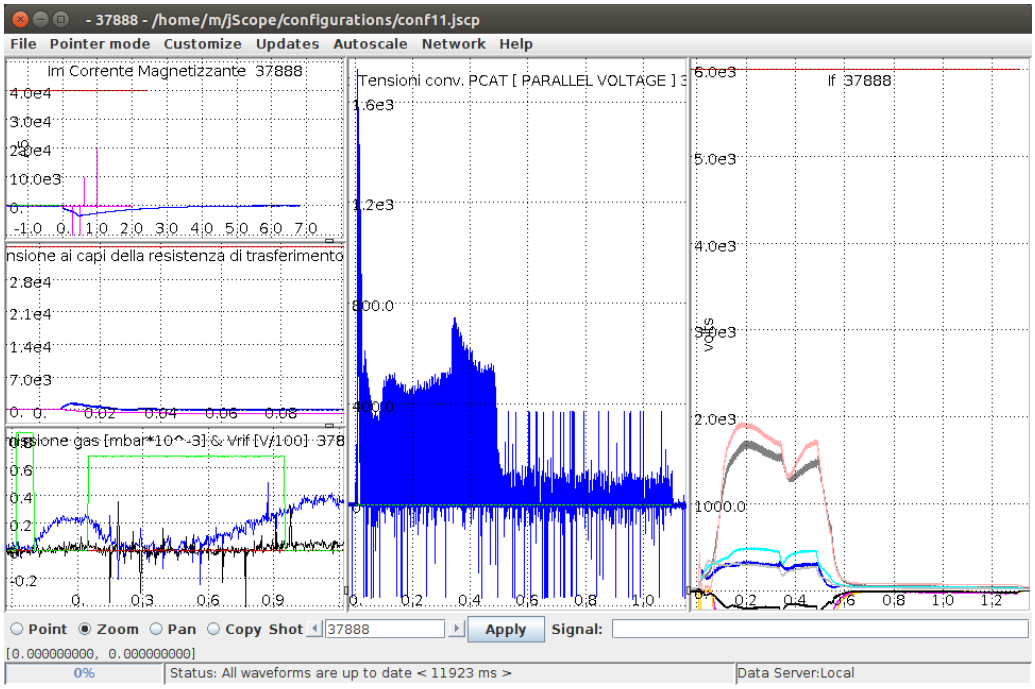


Figura 2.4: Interfaccia di jScope

Capitolo 3

Tecnologie utilizzate

In questo capitolo vengono presentate le tecnologie usate nello sviluppo di WebScope.

3.1 JavaScript e AJAX

JavaScript è un linguaggio di programmazione usato in modo predominante all'interno dei browser web per consentire l'esecuzione di script lato client e la creazione di applicazioni web. Negli ultimi anni si è inoltre avuta una diffusione del suo utilizzo anche nell'ambito della programmazione lato server e della realizzazione di applicazioni mobili. Sviluppato inizialmente da Brendan Eich per il browser Netscape e standardizzato nel 1997 con il nome ECMAScript, è un linguaggio interpretato, con tipizzazione dinamica e supporta la programmazione orientata agli oggetti. Pur essendo un linguaggio *general purpose*, molte delle sue caratteristiche sono volte a semplificare le operazioni che riguardano l'esecuzione all'interno di un browser, come la creazione e al modifica di elementi HTML e la gestione degli input dell'utente.

AJAX, acronimo di Asynchronous JavaScript And XML, è una metodologia di comunicazione tra le componenti client e server di un'applicazione web, che utilizza una combinazione di tecnologie (JavaScript, DOM, HTML, XML, JSON) per consentire di effettuare richieste e ricevere risposte in modo asincrono. L'interazione standard tra un browser e un server web prevede l'invio di richieste da parte del browser per ottenere una pagina web e le risorse collegate, come le immagini, e quando i dati vengono ricevuti la pagina viene visualizzata. Questo approccio presenta due inconvenienti particolarmente rilevanti se usato nel caso di applicazioni web e non semplici documenti: dal momento in cui vengono effettuate le richieste fino alla ricezione dei dati l'utente non può interagire con la pagina visualizzata, e per aggiornare anche solo una parte della pagina è necessario ricaricarla interamente del server, causando un'attesa per l'utente e una ridondanza nelle comunicazioni.

AJAX fornisce un meccanismo di comunicazione che permette al client di effettuare richieste usando un oggetto JavaScript chiamato `XMLHttpRequest`, senza che sia necessario ricaricare la pagina, e di usare i dati ricevuti per modificare dinamicamente il contenuto della stessa. Una richiesta viene creata con la funzione `open` dell'oggetto

`XMLHttpRequest`, specificando l'URL e il metodo HTTP da usare (GET, POST, ecc.), e viene inviata con la funzione `send`. La risposta viene poi letta da una funzione di callback che viene definita al momento della creazione della richiesta, e nel frattempo possono essere eseguite altre parti di codice, consentendo all'utente di continuare a interagire con l'applicazione. In figura 3.1 è mostrato un confronto tra i due modelli di comunicazione, quello classico e quello basato su AJAX. Nonostante il nome, in AJAX sono supportati altri formati per lo scambio dei dati oltre a XML, come HTML, JSON, stringhe di testo semplici e dati binari.

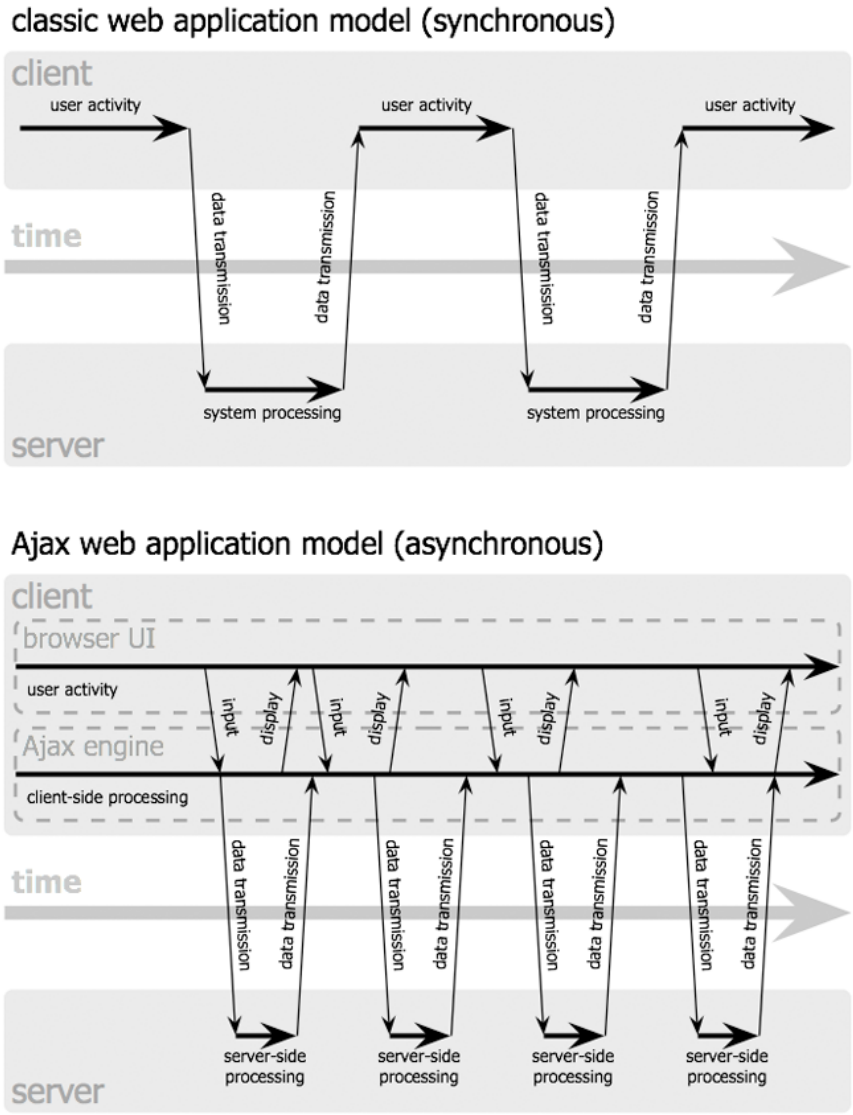


Figura 3.1: Confronto tra il modello di comunicazione classico e quello basato su AJAX. Fonte: [9]

3.2 SVG

SVG (Scalable Vector Graphics) è un linguaggio basato su XML per la descrizione di elementi grafici vettoriali mediante l'uso di primitive geometriche come punti, linee, curve e poligoni. L'utilizzo di un formato vettoriale come SVG presenta diversi vantaggi rispetto a formati grafici raster come JPEG e PNG, in cui invece l'immagine è rappresentata da un insieme di punti (pixel):

- gli elementi grafici possono essere scalati in dimensione senza incorrere in problemi di perdita di qualità.
- l'immagine risultante viene generata lato client dal browser, sulla base di una descrizione testuale inviata dal server (SVG, essendo basato su XML, è un formato testuale e non binario), riducendo considerevolmente l'occupazione di banda e di conseguenza il tempo necessario al trasferimento.
- gli elementi grafici possono essere modificati dal client, ad esempio attraverso JavaScript, senza che sia necessaria alcuna comunicazione con il server.

Gli elementi grafici che costituiscono un'immagine in formato SVG sono organizzati in una struttura ad albero, con un elemento radice, `<svg>`, e numerosi possibili tipi di elementi che rappresentano le componenti grafiche di cui l'immagine è composta. Ogni elemento presenta degli attributi con valori numerici, come la posizione e la dimensione, che sono riferiti ad un sistema di coordinate avente l'origine nell'angolo in alto a sinistra dell'immagine.

Gli elementi usati per rappresentare le forme geometriche di base sono `<rect>` per i rettangoli, `<line>` per i segmenti, `<circle>` per i cerchi, `<ellipse>` per le ellissi, `<polyline>` per le linee spezzate e `<polygon>` per i poligoni. Gli ultimi due sono caratterizzati da un attributo `points` che definisce la lista dei vertici della linea spezzata o del poligono, indicando le coordinate di ciascuno di essi. Per generare una forma di qualsiasi tipo, comprese le linee curve, è disponibile l'elemento `<path>`, il cui attributo `d` contiene la descrizione della forma attraverso l'uso di un insieme di comandi e parametri. Ogni comando è indicato da una lettera seguita un certo numero (dipendente dal tipo di comando) di parametri numerici, ad esempio il comando "`l 10,20`" specifica di tracciare un segmento dal punto corrente al punto che si trova 10 pixel verso destra e 20 pixel verso il basso. Sono disponibili comandi per definire il punto di partenza (`m`), tracciare un segmento (`l`), un arco di ellisse (`a`), una curva di Bézier quadratica (`q`) o cubica (`c`).

Di seguito viene riportato un esempio di codice SVG con i diversi elementi descritti, e la corrispondente immagine è mostrata in figura 3.2.

```
1 <?xml version="1.0"?>
2 <svg xmlns="http://www.w3.org/2000/svg" width="350" height="150">
3   <rect x="15" y="15" width="60" height="29" fill="red"
        stroke="black"/>
4   <line x1="127" y1="8" x2="104" y2="44" fill="none" stroke="blue"
        stroke-width="2"/>
5   <circle r="20" cx="190" cy="33" fill="green" stroke="black"/>
```

```
6     <polyline points="256,49 275,19 277,47 317,17 350,50" fill="none"
          stroke="orange" stroke-width="2"/>
7     <polygon points="11,105 32,65 70,82 69,103 34,91 28,108"
          fill="yellow" stroke="black"/>
8     <path d="m 111,98
9              l 30,-5
10             c 8,-19 38,-1 38,-1
11             c 0,0 -66,39 8,7
12             c 74,-32 21,-39 74,-32
13             c 53,7 -19,-12 37,18
14             q 56,30 58,30" stroke-width="2" fill="none" stroke="black"/>
15  </svg>
```

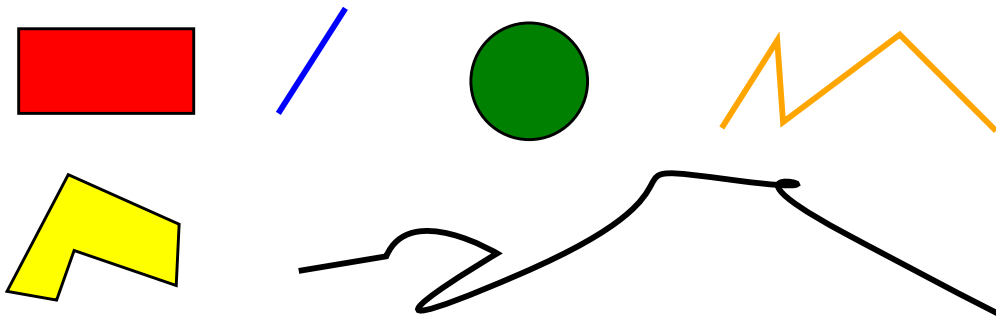


Figura 3.2: Esempio di immagine SVG

Attraverso l'uso di attributi possono essere modificate le proprietà degli elementi SVG, come i colori da usare per i contorni e il riempimento, lo spessore delle linee, il grado di trasparenza e trasformazioni geometriche come rotazioni, traslazioni e cambi di scala. È possibile creare gradienti e pattern di colore da applicare agli elementi, e filtri che permettono di ottenere effetti come sfocature e distorsioni.

3.3 Canvas

Canvas è un elemento HTML che rappresenta una superficie su cui è possibile disegnare elementi grafici e immagini usando il linguaggio JavaScript. È stato introdotto per la prima volta nel 2004 da Apple all'interno del motore di rendering WebKit, e successivamente è stato standardizzato come parte di HTML5.

A differenza di SVG, che è un formato vettoriale, tutte le forme grafiche e le immagini all'interno di un elemento canvas sono disegnate come bitmap, pertanto non è possibile modificare una parte di un elemento canvas o ridimensionare la superficie da esso occupata senza ridisegnarlo interamente. Per esempio, in SVG si può inserire un elemento `<circle>` per visualizzare un cerchio e successivamente ridimensionarlo attraverso la modifica dei suoi attributi, senza dover tenere in considerazione eventuali altri elementi grafici presenti; in un elemento canvas, una volta disegnato un cerchio non è

possibile fare riferimento ad esso come entità distinta dalle altre forme grafiche, e volendo in seguito ridimensionarlo è necessario cancellare tutta la superficie dell'elemento canvas e disegnare il cerchio con la nuova dimensione e tutti gli altri elementi grafici che erano presenti[1]. SVG e canvas possono essere considerate due tecnologie complementari, ciascuna adatta a diverse situazioni di utilizzo.

Un elemento canvas è caratterizzato dagli attributi `width` e `height` che definiscono la dimensione della superficie, e ad esso è associato un *rendering context* che viene usato per eseguire le operazioni di creazione degli elementi grafici. Il *rendering context* è rappresentato in JavaScript da un oggetto `CanvasRenderingContext2D`, che mette a disposizione numerose funzioni per disegnare forme grafiche e modificare le proprietà dell'elemento canvas.

Le primitive geometriche supportate dal *rendering context* sono i rettangoli e i percorsi (*path*), e come in SVG viene usato un sistema di coordinate con l'origine nell'angolo in alto a sinistra dell'elemento canvas. Per disegnare un rettangolo sono disponibili le funzioni `strokeRect` (che disegna solo il perimetro) e `fillRect` (che colora anche l'area interna), mentre la funzione `clearRect` viene usata per cancellare un'area dell'elemento canvas. I percorsi sono costituiti da una sequenza di segmenti e linee curve, che possono avere dimensioni e colori diversi fra loro. Per disegnare un percorso si invoca la funzione `beginPath`, quindi una o più funzioni per definire le varie componenti del percorso, e infine si dichiara completato il percorso con `closePath` e lo si disegna effettivamente con le funzioni `stroke`, per disegnare solo la linea del percorso, o `fill`, per colorare l'area da esso racchiusa. Le funzioni disponibili per definire le componenti del percorso sono `moveTo`, che specifica il punto iniziale, `lineTo` per le linee, `arcTo` per gli archi, `ellipse` per tratti di ellisse, `bezierCurveTo` e `quadraticCurveTo` per curve di Bézier cubiche e quadratiche. Sia per i rettangoli che per i percorsi vengono usate le funzioni `strokeStyle` e `fillStyle` per impostare i colori da usare per i perimetri e le aree.

Altre funzioni sono disponibili per definire spessori, stili, trasparenze, ombreggiature e gradienti di colore degli elementi geometrici e per creare elementi testuali.

Oltre alle forme geometriche è possibile inserire in un elemento canvas delle immagini, ricavate da un elemento HTML `<img>`, da un frame di un elemento `<video>` o da un altro elemento `<canvas>`. Una volta ottenuto un riferimento all'immagine da utilizzare, viene usata la funzione `drawImage` del *rendering context* per inserirla nell'elemento canvas. La funzione presenta tre varianti: la più semplice inserisce l'immagine alle coordinate specificate come parametri, la seconda permette di scalare l'immagine definendo la larghezza e l'altezza che dovrà avere all'interno dell'elemento canvas, e la terza consente di ritagliare una porzione dell'immagine di partenza ed eventualmente scalarla.

I pixel che costituiscono l'elemento canvas sono rappresentati nel loro insieme da un oggetto `ImageData`, che ha come attributi la larghezza e l'altezza dell'elemento canvas

[1] In alcuni casi può essere sufficiente ridisegnare solo una parte dell'elemento canvas, quella corrispondente all'area dove si trova l'elemento grafico che si vuole modificare, ma questo implica dover individuare quali altri elementi si sovrappongono ad esso per poterli poi ridisegnare, e il procedimento può risultare quindi più complesso.

e un array di byte che, a gruppi di quattro, rappresentano i valori dei canali *red*, *green*, *blue* e *alpha* di ciascun pixel. Un oggetto `ImageData` può essere ricavato da un elemento canvas esistente con la funzione `getImageData` del *rendering context*, oppure può essere creato da zero con `createImageData`, precisando le dimensioni volute. Si possono quindi modificare i valori dei singoli pixel e usare l'oggetto `ImageData` risultante per aggiornare il contenuto di un elemento canvas con la funzione `putImageData`.

3.4 Python e WSGI

Python è un linguaggio di programmazione creato nel 1989 da Guido van Rossum, che si distingue per la leggibilità, la compattezza della sintassi e la facilità di utilizzo rispetto a linguaggi come C++ e Java. È un linguaggio con tipizzazione dinamica che supporta la programmazione orientata agli oggetti e la programmazione funzionale, dispone di un'ampia libreria standard che supporta tutte le più comuni operazioni ed è estensibile grazie alla possibilità di includere moduli scritti in altri linguaggi come C e C++, per ottenere quando necessario una velocità di esecuzione maggiore o l'accesso a funzionalità di basso livello.

WSGI (Web Server Gateway Interface) è uno standard che definisce l'interfaccia tra un server web e un'applicazione o un framework Python, sviluppato per consentire di realizzare applicazioni web eseguibili senza necessità di modifiche su qualsiasi server web che supporta lo standard, come Apache e Nginx.

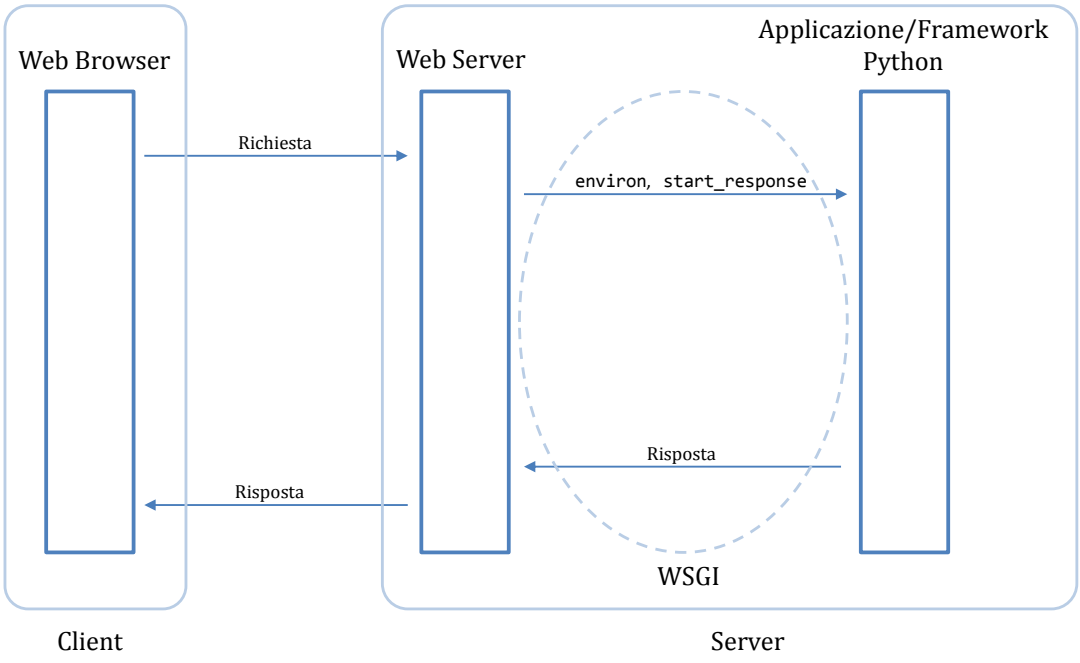


Figura 3.3: Schema dell'interfaccia WSGI

WSGI prevede che, per ogni richiesta HTTP ricevuta da un client, il server fornisca alla classe che rappresenta l'applicazione web una struttura dati (`environ`) e una funzione di callback (`start_response`). La struttura dati `environ` contiene informazioni come il metodo HTTP usato nella richiesta (GET, POST, ecc.), l'URL richiesto, la *query string* e impostazioni di configurazione come la posizione del file in cui l'applicazione può registrare eventuali messaggi di errore. Sulla base di questi dati l'applicazione elabora la richiesta e restituisce al server la risposta da inviare al client, invocando prima la funzione `start_response` con due parametri: il codice di stato HTTP e una lista di coppie di stringhe che rappresentano i nomi e i valori degli header HTTP.

Capitolo 4

WebScope

In questo capitolo viene descritta nel dettaglio la versione iniziale di WebScope, presentando l'architettura e le funzionalità messe a disposizione dall'applicazione e i suoi particolari implementativi.

4.1 Architettura e funzionalità

WebScope è un'applicazione web con un'architettura di tipo client-server, costituita da una componente server, realizzata in Python, che si interfaccia con il sistema MDSplus per prelevare i dati e gestire le richieste della componente client. Quest'ultima, realizzata in JavaScript, una volta ottenuti i dati necessari, si occupa della loro visualizzazione e della gestione delle interazioni con l'utente.

L'applicazione può essere avviata visitando l'indirizzo `http://<host>/mdsplusWsgi/scope?user=<username>`, dove `<host>` è il nome o l'indirizzo IP del server su cui è installato WebScope, mentre `<username>` è il nome dell'utente ai cui file di configurazione si vuole accedere. La pagina che viene visualizzata contiene una lista dei file di configurazione disponibili. I file di configurazione, creati da jScope, specificano sia quali informazioni si vogliono visualizzare, ad esempio le espressioni dei segnali, sia le modalità di presentazione, come il numero, le dimensioni e la disposizione dei pannelli in cui è suddivisa l'area di visualizzazione, il titolo di ciascun pannello e le etichette degli assi. I file di configurazione sono descritti nel paragrafo 4.2.1.

Selezionando un file di configurazione dalla lista viene aperta una nuova scheda nel browser e caricata la componente client dell'applicazione che, tramite AJAX, richiede al server il contenuto del file di configurazione. Quest'ultimo viene prima convertito in formato XML dal server, in modo da rendere più semplice il *parsing* in JavaScript rispetto al formato nativo di jScope, e quindi inviato al client, che può a questo punto creare i pannelli in cui saranno visualizzati i segnali e, per ciascun pannello, richiedere al server i dati necessari alla visualizzazione dei segnali e degli altri elementi in esso contenuti. Ricevuta la risposta dal server, il client genera, all'interno di ogni pannello, gli elementi grafici SVG corrispondenti ai segnali, le griglie, i titoli e le etichette degli assi.

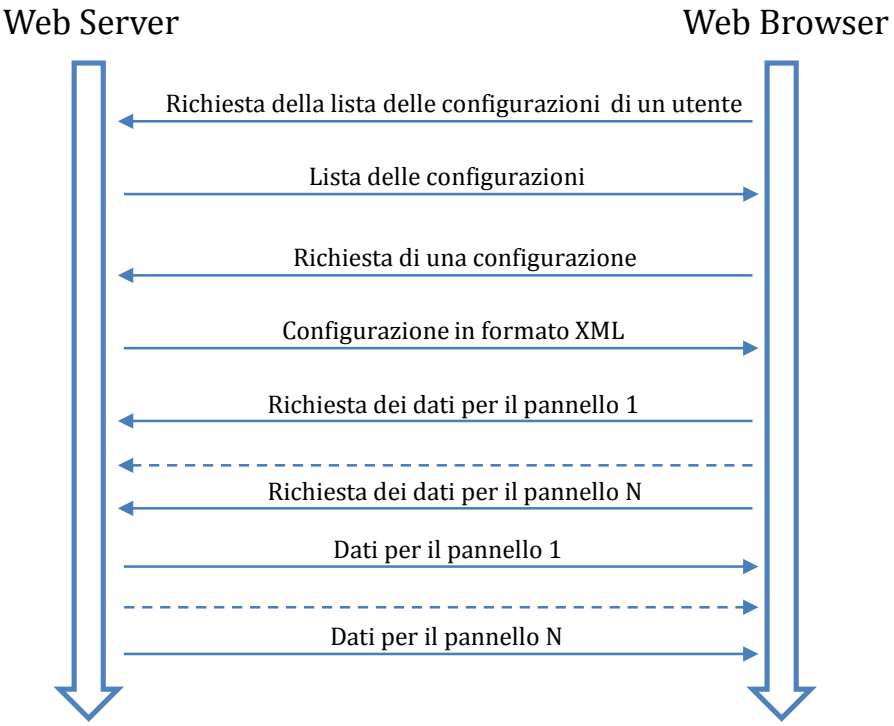


Figura 4.1: Comunicazione client-server all'avvio dell'applicazione. Fonte: adattata da [1]

La figura 4.2 mostra uno screenshot dell'applicazione una volta completato il caricamento. La maggior parte della pagina è occupata dai pannelli con i segnali, mentre nell'area inferiore sono presenti i controlli che permettono all'utente di selezionare una delle tre modalità di interazione: zoom, pan e crosshair. A seconda della modalità selezionata, l'utente può effettuare diverse operazioni:

- in modalità zoom può selezionare una porzione di segnale da ingrandire, tracciando con il mouse un rettangolo intorno ad essa.
- in modalità pan può traslare i segnali in modo da cambiare la porzione visualizzata, con un click-and-drag in un punto qualsiasi del pannello.
- in modalità crosshair, facendo click su un punto di un segnale può ottenere il valore del segnale stesso in quel punto, nonché l'espressione del segnale, informazioni che vengono visualizzate nell'area in basso a destra della schermata. Inoltre in tutti i pannelli vengono visualizzati dei crosshair in corrispondenza dell'ascissa del punto selezionato, cosa che può essere utile per evidenziare i valori assunti dall'insieme dei segnali in un certo istante.

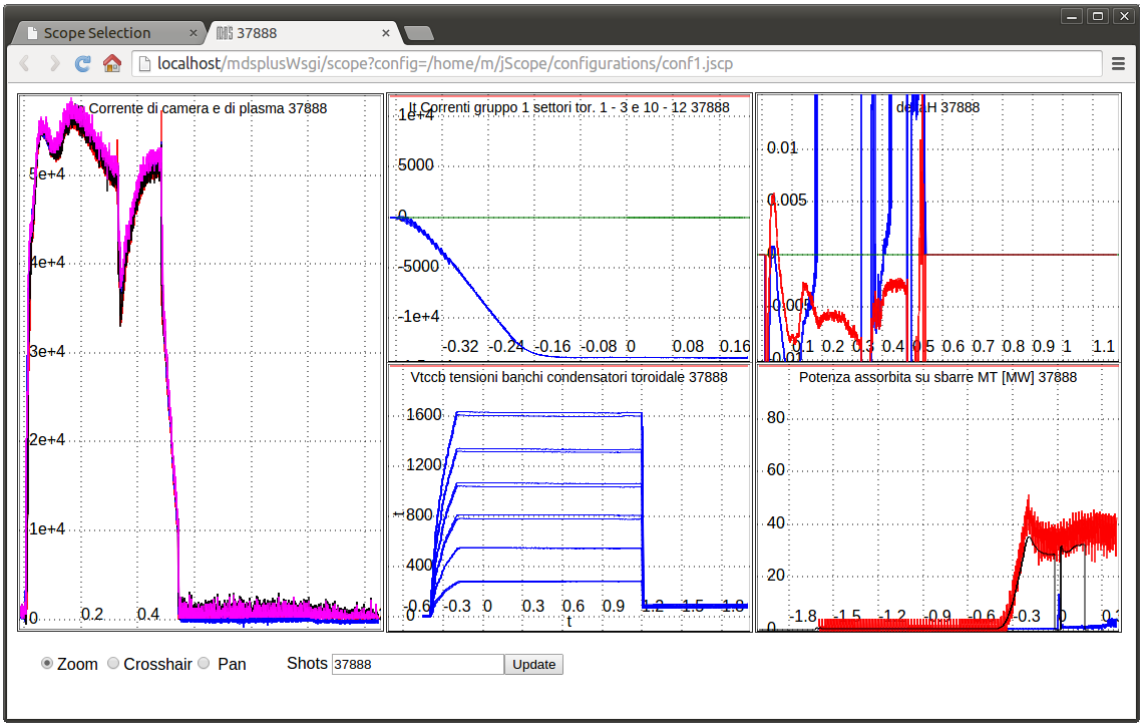


Figura 4.2: Interfaccia di WebScope

Il campo "shots" consente di specificare il numero dello *shot* da visualizzare; nel caso ne vengano indicati più di uno verranno mostrate, per ciascun segnale, le sue varie versioni appartenenti ai diversi *shot*. Facendo clic con il tasto destro del mouse su un pannello si accede ad un menù con vari pulsanti per modificare la scala con cui vengono visualizzati i segnali. Le opzioni disponibili sono:

- Autoscale: visualizza i segnali del pannello nella loro interezza, ossia nell'intervallo compreso tra la minima e la massima ascissa (in orizzontale) e tra la minima e la massima ordinata (in verticale). Da notare che, nel caso un pannello contenga più segnali, essi possono avere ascisse minime e massime diverse.

- Autoscale Y: ha lo stesso comportamento di Autoscale ma lascia invariato l'intervallo delle ascisse visualizzato.

- Autoscale All: applica Autoscale a tutti i pannelli.

- All same scale: imposta gli intervalli visualizzati di tutti i pannelli a quelli del pannello selezionato, ovvero, se il pannello selezionato visualizza gli intervalli $[x_0, x_1]$ delle ascisse e $[y_0, y_1]$ delle ordinate, tutti i pannelli visualizzeranno gli intervalli $[x_0, x_1]$ e $[y_0, y_1]$ dei segnali in essi contenuti.

- All same X scale: imposta gli intervalli delle ascisse visualizzati di tutti i pannelli a quello del pannello selezionato, lasciando invariati gli intervalli delle ordinate.
- All same X scale auto Y: detto $[x_0, x_1]$ l'intervallo delle ascisse del pannello selezionato, imposta gli intervalli delle ascisse visualizzati di tutti i pannelli a $[x_0, x_1]$, e gli intervalli delle ordinate al minimo e massimo delle ordinate dei segnali nell'intervallo $[x_0, x_1]$.
- All same Y scale: imposta gli intervalli delle ordinate visualizzati di tutti i pannelli a quello del pannello selezionato, lasciando invariati gli intervalli delle ascisse.
- Reset scales: nel file di configurazione è possibile specificare, per ciascun pannello, degli intervalli predefiniti di ascisse e ordinate da visualizzare. Selezionando Reset scales gli intervalli visualizzati vengono reimpostati a quelli predefiniti.
- Reset all scales: applica Reset scales a tutti i pannelli.

Oltre all'interazione tramite mouse, WebScope supporta i dispositivi touchscreen e permette quindi di effettuare tutte le operazioni descritte anche su smartphone e tablet. Infine possono essere gestiti gli eventi MDSplus, facendo in modo che il server rimanga in attesa di un certo evento e notifichi il client quando questo si verifica, il quale potrà ad esempio richiedere l'aggiornamento dei dati relativi ad un pannello.

4.2 Implementazione lato server

La componente server di WebScope è implementata in linguaggio Python impiegando l'interfaccia WSGI, e in particolare viene usato il modulo mod_wsgi che consente di eseguire applicazioni web WSGI su server Apache HTTP. Come per tutte le applicazioni WSGI, viene definita una classe `application` che riceve in ingresso l'URL richiesto e la *query string* con l'insieme dei parametri passati dal client, nella forma `/tipo-richiesta/parametro1/parametro2/.../parametroN?opzione1=valore1 &opzione2=valore2&...` . A seconda del tipo di richiesta specificato viene richiamata una delle funzioni della classe `application`, che legge i parametri in ingresso e utilizza l'interfaccia Python di MDSplus per ricavare i dati richiesti, i quali vengono quindi restituiti nella risposta al client. I possibili tipi di richieste sono descritti di seguito.

4.2.1 Il tipo di richiesta `scope`

Il tipo di richiesta principale è `scope`. In base alle opzioni presenti nella *query string* si può suddividere in quattro varianti: `user`, `configxml`, `panel` e `title`. Se l'URL è nella forma `/scope?user=<username>[&dir=<percorso>]`, come visto nella sezione 4.1, viene letta dal file system la lista dei file di configurazione (cercandoli nella percorso indicato dall'opzione `dir` oppure, se assente, in una posizione predefinita), e restituita al client una pagina HTML contenente un link per ciascun file di configurazione. La seconda variante, nella forma `/scope?configxml=<nome file di configurazione>`,

viene inviata quando l'utente seleziona uno dei link: il file di configurazione specificato viene letto, convertito in formato XML e trasmesso al client. Un esempio di file di configurazione in versione XML è mostrato di seguito:

```
1  <?xml version="1.0"?>
2  <scope>
3    <title><expression>"'Configurazione di esempio'"</expression></title>
4    <palette>
5      <color>Black</color>
6      <color>Blue</color>
7      <color>Cyan</color>
8      <color>Green</color>
9      <color>Magenta</color>
10     <color>Orange</color>
11     <color>Pink</color>
12     <color>Red</color>
13     <color>Yellow</color>
14   </palette>
15   <columns>
16     <column>
17       <panel tree="test_tree" shot="1500" xmin="0.2" xmax="0.8"
18           ymin="0." ymax="1." title="'Pannello 1'"
19           xlabel="'Etichetta ascisse'" ylabel="'Etichetta ordinate'">
20         <signal color="2" mode="3">\test_tree::top:signal1</signal>
21       </panel>
22       <panel tree="test_tree" shot="1500" event="Evento1"
23           title="\test_tree::top:string2">
24         <signal color="5">
25           \test_tree::top:signal2-\test_tree::top:signal4
26         </signal>
27       </panel>
28     </column>
29     <column>
30       <panel tree="test_tree" shot="1500">
31         <signal color="4" mode="3">\test_tree::top:signal3</signal>
32         <signal color="2">\test_tree::top:signal4</signal>
33       </panel>
34     </column>
35   </columns>
36 </scope>
```

In esso viene definito il titolo della configurazione, che viene mostrato come titolo della pagina web, una palette dei colori che possono essere usati per le rappresentazioni grafiche dei segnali e il layout dell'area di visualizzazione, costituito da una o più colonne contenenti ciascuna un numero variabile di pannelli: nell'esempio sono presenti due colonne, la prima con due pannelli e la seconda con uno. Ogni pannello possiede un insieme di attributi, tutti facoltativi a parte i primi due: il `tree` e lo `shot` in cui sono memorizzati i segnali, gli estremi predefiniti di ascisse (`xmin` e `xmax`) e di ordinate (`ymin` e `ymax`) da visualizzare, il titolo del pannello (`title`), le etichette degli assi (`xlabel` e `ylabel`) e il nome dell'evento da gestire (`event`). Nell'esempio il valore dell'attributo `title` del primo pannello è definito direttamente come stringa (riga 18), mentre quello

del secondo pannello è un riferimento ad un nodo del *tree* `test_tree` (riga 23), che contiene la stringa da utilizzare: in generale, i valori degli attributi dei titoli, delle etichette e degli estremi di visualizzazione sono costituiti da espressioni MDSplus, di cui il client, una volta ottenuto il file XML, dovrà richiedere la valutazione inviando una apposita richiesta al server (la variante `panel` descritta in seguito), in modo da ottenere gli effettivi valori (numeri e stringhe) da utilizzare. All'interno di ciascun pannello sono definiti uno o più segnali, con le loro espressioni, e i possibili attributi `color` e `mode`, ossia l'indice del colore nella palette e una delle tre modalità di visualizzazione: linea continua, punti e misto punto-linea, mostrate in figura 4.3.

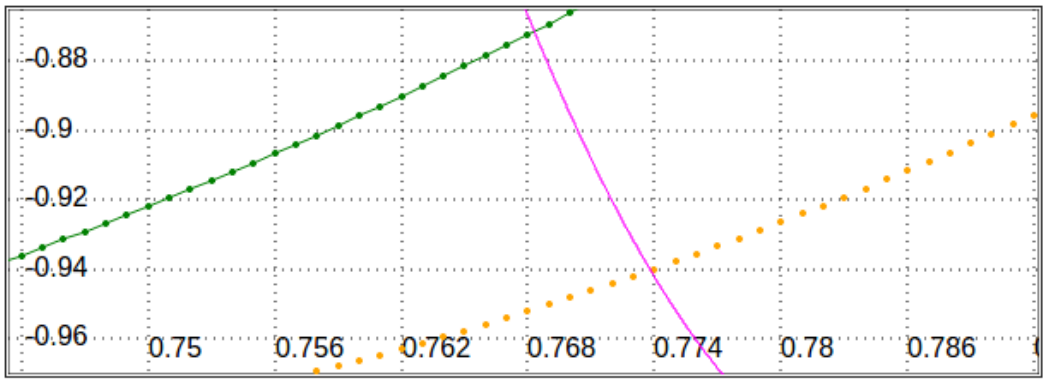


Figura 4.3: Le tre modalità di visualizzazione dei segnali

La variante `panel` viene usata dal client per richiedere i dati necessari alla costruzione di tutti gli elementi grafici di un pannello; ha la forma `/scope?panel=yes&<opzione1>=<espressione1>&<opzione2>=<espressione2>&...`, dove le possibili opzioni sono `title`, `xlabel`, `ylabel`, `xmin`, `xmax`, `ymin`, `ymax`, `default_node`, `tree`, `shot`, `y1`, `x1`, `y2`, `x2`, ecc., tutte facoltative tranne `y1`. L'opzione `default_node` indica l'espressione del nodo che viene usato come punto di partenza nella risoluzione dei percorsi relativi nel *tree* (al contrario dei percorsi assoluti che partono invece dalla radice), mentre alle opzioni `xi` e `yi` sono associate le espressioni delle componenti x e y del segnale i-esimo, e nel caso l'opzione `xi` sia omessa viene usata come componente x la dimensione del segnale (vedi sezione 2.3); infine l'opzione `shot` indica uno o più *shot*, separati da virgole.

Le espressioni ricevute vengono valutate dal server, ricavando per ciascuno *shot* i valori delle componenti x e y degli N segnali, che vengono inviati uno di seguito all'altro come sequenza di byte nella risposta. Negli header HTTP vengono indicati, per ciascun segnale, la lunghezza e il tipo di dato MDSplus (come `Int8`, `Int32` o `Float32`) delle sue componenti, in modo che il client possa costruire una rappresentazione dei segnali a partire dalla sequenza di byte. Altri header sono usati per i valori delle rimanenti espressioni (il titolo, le etichette e gli estremi di visualizzazione), nonché per segnalare eventuali errori.

L'ultima variante, `title`, con la forma `/scope?title=<espressione>`, permette di ottenere il valore del titolo della configurazione.

4.2.2 Il tipo di richiesta `event`

Il tipo di richiesta `event` viene inviato dal client per ogni pannello che ha nel file di configurazione l'attributo `event` definito, e l'URL usato è `/event/<nome-evento>[?timeout=<timeout-in-secondi>&handler=<modulo-handler>]`. Quando il server riceve la richiesta si mette in attesa dell'evento il cui nome è indicato nell'URL, fino allo scadere del tempo stabilito dall'opzione `timeout`, se presente, oppure per un tempo predefinito (un minuto). Se il periodo di timeout scade e l'evento non si è verificato, il client viene notificato del fatto, altrimenti l'evento viene passato come parametro alla funzione handler nel modulo specificato, che viene importato dinamicamente usando il valore dell'opzione `handler`, o, in sua assenza, alla funzione handler predefinita.

Un handler è una funzione Python che riceve un oggetto che rappresenta l'evento, in modo da poterne ricavare i dati associati, e restituisce l'output da inviare come risposta al client, che può essere costituito da informazioni da visualizzare così come da una segnalazione di errore, insieme ad eventuali header HTTP. È quindi possibile per un utente implementare un handler personalizzato in modo da ottenere il comportamento voluto. Se nessun handler viene indicato, quello predefinito restituisce del codice XML nel formato:

```
<event>
    <name>Nome evento</name>
    <time>Istante in cui l'evento si è verificato</time>
    <data>
        <data_bytes>[byte1,byte2,byte3,....,byteN]</data_bytes>
        <data_text>Rappresentazione testuale dei byte</data_text>
    </data>
</event>
```

dove i valori contenuti nei tag `time` e `data` sono ricavati dall'oggetto che rappresenta l'evento.

4.3 Implementazione lato client

La componente client di WebScope è realizzata in JavaScript ed è costituita dalle cinque classi `WavePanel`, `Wave`, `Grid`, `Signal` e `Metrics`, un insieme di funzioni di inizializzazione e uno di funzioni per la gestione degli input. La figura 4.4 mostra le relazioni che sussistono tra le cinque classi, indicando con una freccia da una classe ad un'altra il fatto che la prima contiene un riferimento alla seconda.

4.3.1 Funzioni di inizializzazione e gestione degli eventi

Nel momento in cui la componente client viene caricata, essa invia al server una richiesta tramite AJAX per ottenere il file di configurazione in formato XML. Il *parsing* di quest'ultimo viene eseguito da una funzione che crea un elemento HTML `<table>` rappresentante l'area di visualizzazione dei pannelli, e inserisce al suo interno degli elementi `<svg>`, ciascuno corrispondente ad un pannello, organizzati in colonne come descritto nel file di configurazione.

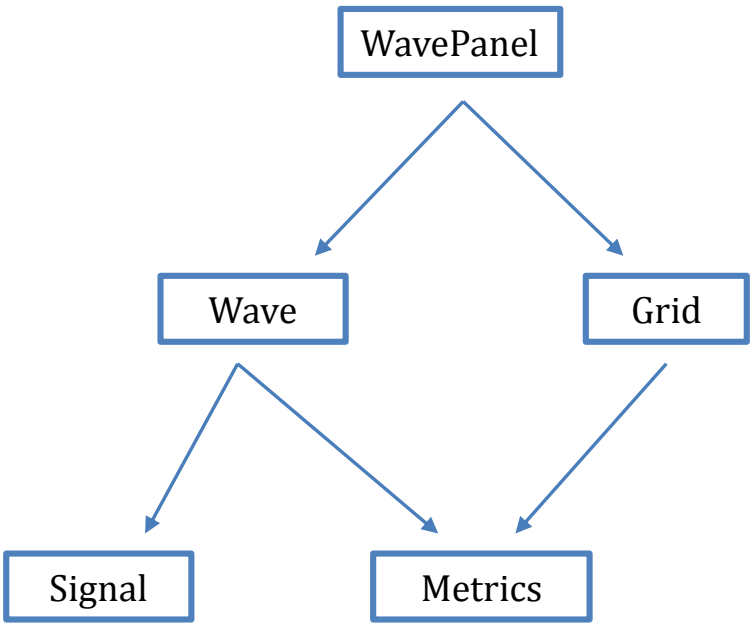


Figura 4.4: Relazioni tra le classi della componente client

Gli attributi di ogni pannello e dei segnali che ad esso appartengono, ricavati dal file di configurazione, vengono utilizzati per creare un oggetto `WavePanel`, e se è presente l'attributo `event` viene effettuata, sempre mediante AJAX, una corrispondente richiesta di tipo `event` al server. L'invio delle richieste `event` viene gestito da una apposita funzione che in seguito rimane in attesa della risposta, la quale può rappresentare o la notifica che il timeout per l'evento è scaduto, o quella che l'evento si è verificato, nel qual caso viene invocata la funzione `update` dell'oggetto `WavePanel` per aggiornare i dati del pannello; successivamente la funzione invia una nuova richiesta `event` e ritorna in uno stato di attesa.

4.3.2 Funzioni per la gestione degli input

Gli eventi JavaScript `mouseDown`, `mouseMove`, `mouseUp`, `touchStart`, `touchMove` e `touchEnd` sono gestiti da omonime funzioni che individuano il pannello su cui si è verificato l'evento e, in base alla modalità selezionata dall'utente, invocano una delle funzioni del corrispondente oggetto `WavePanel`, descritte più avanti. Ad esempio, se la modalità selezionata è "pan", al verificarsi di un evento `mouseDown` verrà invocata la funzione `startPan` dell'oggetto `WavePanel`. Se l'evento corrisponde ad un click con il tasto destro del mouse, viene invece visualizzato il menù con le varie opzioni di scaling. In modo

simile altre funzioni si occupano di richiedere agli oggetti `WavePanel` l'aggiornamento della visualizzazione degli elementi grafici quando viene ridimensionata la finestra del browser e quando l'utente seleziona una delle opzioni di scaling dal relativo menù.

4.3.3 La classe `Signal`

La classe `Signal` rappresenta un segnale, costituito dalle sue componenti x e y, memorizzate come array in modo tale che le coordinate del punto i-esimo del segnale siano contenute in `x[i]` e `y[i]`. Il tipo di ciascuna componente può essere uno qualunque dei tipi di dato supportati da MDSplus, ognuno dei quali viene mappato in un tipo di array JavaScript, come `Int32Array` o `Float64Array`. La classe mantiene inoltre le informazioni sul colore da usare nella rappresentazione segnale, la modalità di visualizzazione (linea continua, punti o misto punto-linea), l'espressione MDSplus del segnale e lo *shot* a cui appartiene. Sono poi presenti delle funzioni per ottenere il minimo e il massimo delle componenti x e y in tutto il segnale o in un intervallo di ascisse (cosa necessaria per implementare le varie funzionalità di scaling), e per ottenere il punto del segnale più vicino ad un dato punto del pannello, nonché la distanza tra i due (per la gestione della funzionalità di crosshair).

4.3.4 La classe `Metrics`

Tutte le conversioni tra le coordinate dei punti dei segnali e le coordinate dei pixel all'interno dell'area di un pannello, definita da un elemento `<svg>`, sono gestite dalla classe `Metrics`. Le variabili su cui si basa la conversione sono la larghezza e l'altezza dell'elemento `<svg>`, i valori minimo e massimo delle ascisse e delle ordinate del segnale (o dell'insieme di segnali) del pannello e l'ampiezza del margine interno del pannello[1]. In base ad esse viene calcolato un fattore di conversione che viene poi utilizzato dalle funzioni della classe per ottenere le coordinate di un pixel a partire da quelle di un punto del segnale e viceversa.

4.3.5 La classe `Wave`

La classe `Wave` rappresenta l'insieme dei segnali visualizzati in un pannello, memorizzati come array di oggetti `Signal`. La funzione principale messa a disposizione dalla classe è `plot`, la quale, partendo dagli oggetti `Signal`, produce gli elementi SVG che costituiscono le rappresentazioni grafiche dei segnali, sfruttando la classe `Metrics` per le conversioni tra valori dei segnali e coordinate dei pixel. In particolare viene usato un elemento `<path>`, un insieme di elementi `<circle>` o una combinazione dei due per rappresentare rispettivamente un segnale in modalità linea continua, punti o misto punto-linea. Sono inoltre presenti delle funzioni di supporto alla gestione delle operazioni di crosshair, che permettono ad esempio di stabilire qual è il segnale più vicino a un punto del pannello su cui l'utente ha fatto click.

[1] Per impostazione predefinita esiste un sottile margine tra il bordo del pannello e l'area in cui sono presenti gli elementi grafici.

4.3.6 La classe `Grid`

Oltre ai segnali un pannello contiene altri elementi grafici, che nel loro insieme sono rappresentati dalla classe `Grid`: la griglia di sfondo, gli indicatori dei valori lungo gli assi, le etichette degli assi e il titolo del pannello. Gli elementi SVG usati sono `<line>` per le linee che compongono la griglia e `<text>` per indicatori, etichette e titolo. Come per la classe `Wave`, è presente una funzione `plot` che genera gli elementi SVG usando la classe `Metrics` per le conversioni.

4.3.7 La classe `WavePanel`

Ogni pannello è rappresentato da un'istanza della classe `WavePanel`, la quale mantiene al suo interno un riferimento ad un oggetto di tipo `Wave` e ad uno di tipo `Grid`, che insieme rappresentano tutti gli elementi grafici del pannello. Le funzioni della classe possono essere suddivise in tre gruppi, che gestiscono rispettivamente l'aggiornamento dei dati del pannello, le operazioni di scaling e di ridimensionamento del pannello, e le funzionalità di zoom, pan e crosshair.

L'aggiornamento dei dati del pannello, ossia i valori delle componenti x e y dei segnali, dei loro attributi e degli attributi del pannello (il titolo, le etichette, ecc.) viene effettuato al momento della creazione dell'oggetto `WavePanel` e, in seguito, ogni volta che si renda necessario a causa del verificarsi di un evento (se previsto tra gli attributi del pannello) o del cambiamento di *shot* tramite il campo "shots" dell'applicazione. La funzione che gestisce l'aggiornamento, `update`, costruisce una richiesta di tipo `scope` nella variante `panel` con i parametri necessari e la invia al server; alla ricezione della risposta, vengono creati gli oggetti `Wave` e `Grid` a partire dai dati ricevuti e ne vengono invocate le funzioni `plot` per generare gli elementi SVG.

Le operazioni di scaling e di ridimensionamento del pannello vengono gestite dalle funzioni `autoscale`, `autoscaleY`, `setScales`, `setScaleAutoY`, `resetScales` e `resize` che, per ottenere l'effetto richiesto, modificano l'oggetto `Metrics` associato agli oggetti `Wave` e `Grid` e invocano le funzioni `plot` di questi ultimi.

Per ciascuna delle funzionalità di zoom, pan e crosshair sono presenti tre funzioni che vengono invocate nel momento in cui si verificano rispettivamente un evento `mouseDown`, `mouseMove` o `mouseUp` (o i corrispondenti eventi touch); ad esempio, per lo zoom le tre funzioni sono `startZoom`, `dragZoom` e `endZoom`. Le operazioni compiute dalle funzioni comprendono la visualizzazione del rettangolo che delimita l'area su cui effettuare uno zoom e delle due linee mostrate durante il crosshair; le funzioni `endZoom` e `dragPan` si occupano poi di modificare l'oggetto `Metrics` e di invocare le funzioni `plot` in maniera simile a quanto fatto dalle funzioni di scaling, in modo da eseguire rispettivamente l'ingrandimento e lo spostamento della regione dei segnali visualizzata.

Capitolo 5

Funzionalità implementate

In questo capitolo vengono descritte le nuove funzionalità implementate in WebScope durante l'attività di tesi partendo dalla versione presentata nel capitolo precedente.

5.1 Gestione del resampling dei segnali

5.1.1 Introduzione dell'uso del resampling

Nella versione iniziale di WebScope, ogni volta che viene richiesto un segnale al server esso viene restituito nella sua interezza, ossia tutti i punti del segnale vengono inclusi nella risposta e utilizzati dal client nella creazione dell'elemento SVG che costituisce la rappresentazione del segnale stesso. Dato che un segnale può essere composto da un numero molto elevato di punti, nell'ordine delle centinaia di migliaia o milioni, questo comporta da un lato un utilizzo di banda relativamente elevato, e quindi un delay nella visualizzazione, dall'altro la necessità da parte del client di gestire elementi SVG molto complessi, con una conseguente diminuzione della reattività agli input dell'utente, particolarmente evidente nelle operazioni di pan.

Per ovviare a questi problemi si è fatto ricorso al resampling "on the fly" lato server, eseguito cioè nel momento stesso in cui un segnale viene richiesto dal client. Il resampling porta ad ottenere un segnale costituito da un numero di punti inferiore rispetto all'originale, ma sufficiente a mantenerne la qualità di visualizzazione. La funzione di MDSplus usata per il resampling è chiamata `GetXYWave`, e richiede come parametri l'espressione del segnale, gli estremi minimo e massimo di ascisse della porzione di segnale che si vuole ottenere e il numero di punti richiesto. La possibilità di richiedere solo una parte del segnale permette di ricevere solo i punti effettivamente necessari, cosa utile nel caso di visualizzazione in modalità zoom.

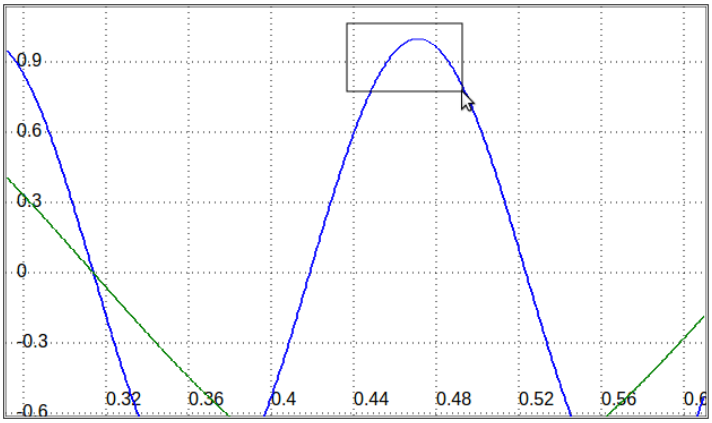
Le modifiche che è stato necessario apportare a WebScope per introdurre la richiesta di resampling sono limitate. A lato client è stato sufficiente cambiare, all'interno della funzione che si occupa dell'aggiornamento dei dati del pannello, i valori delle opzioni `y1`, `y2`, ..., `yn` inviati nella richiesta al server, in modo da introdurre l'uso della funzione `GetXYWave` e i relativi parametri. Questi ultimi variano a seconda del tipo di operazione

che ha richiesto l'aggiornamento, come spiegato in seguito; ad esempio, se la richiesta di aggiornamento è quella che viene eseguita alla creazione del pannello, l'intervallo di ascisse specificato è $[-\infty, \infty]$ (rappresentato in JavaScript da `-Number.MAX_VALUE, Number.MAX_VALUE`), in modo da ottenere una rappresentazione dell'intero segnale, mentre il numero di punti usato è un valore di default, nel seguito indicato con N_{def}, che dipende dalla larghezza dei pannelli, per assicurare una buona qualità di visualizzazione. A lato server non è risultata necessaria alcuna modifica, in quanto la presenza della funzione `GetXYWave` nelle espressioni dei segnali è trasparente al server, che si limita a richiedere, tramite l'interfaccia a oggetti di MDSplus, la loro valutazione, per poi restituire i risultati al client.

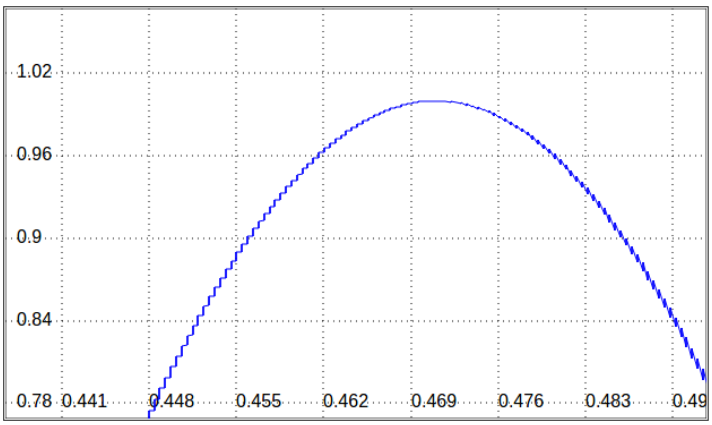
Quello che ha richiesto più attenzione è la modifica del comportamento delle operazioni di zoom e pan in modo da gestire l'introduzione del resampling, ossia il fatto che, al contrario della versione iniziale di WebScope, subito dopo la creazione di un pannello il client non ha a disposizione i segnali in forma completa, ma una loro versione ricampionata. Questo implica che, eseguendo uno zoom su una porzione del segnale, esso apparirà con una qualità insufficiente, dato che, se ad esempio il segnale inizialmente è rappresentato nell'intervallo $[0, 100]$ con 1000 punti, eseguendo uno zoom sull'intervallo $[10, 20]$ verrà visualizzato un elemento SVG composto da 100 punti, e siccome la dimensione del pannello non cambia, la risoluzione si riduce di un fattore 10.

5.1.2 Gestione dello zoom

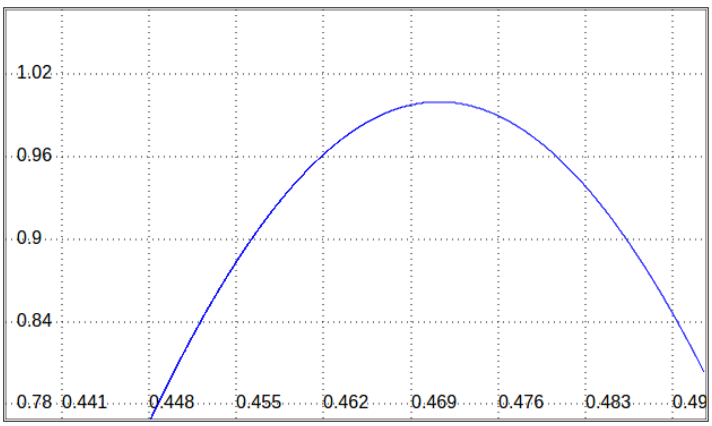
Si rende quindi necessario eseguire una richiesta al server per ogni operazione di zoom, indicando come intervallo di ascisse quello su cui è stato fatto lo zoom, e il valore di default N_{def} per il numero di punti. L'operazione di zoom avviene quindi in questo modo: quando l'utente rilascia il mouse dopo aver tracciato l'area da ingrandire, la funzione `endZoom` della classe `WavePanel`, che viene richiamata come descritto nel paragrafo 4.3.7, richiede l'aggiornamento dei dati dei segnali, e subito dopo, per dare un feedback immediato all'utente sull'operazione compiuta, invoca la funzione `plot` in modo da visualizzare il segnale ingrandito utilizzando i dati correnti (quindi in "bassa risoluzione"). Questo è possibile perché la richiesta di aggiornamento viene portata a termine tramite AJAX, quindi in modo asincrono, e in attesa della risposta il client può effettuare altre operazioni. Non appena viene ricevuta la risposta dal server, una nuova invocazione di `plot` aggiorna i segnali visualizzati con la risoluzione corretta. La figura 5.1 illustra le tre fasi della procedura; in condizioni normali, i segnali in bassa risoluzione vengono visualizzati per una frazione di secondo, corrispondente al *round trip time* della richiesta di aggiornamento.



(a) Selezione dell'area da ingrandire



(b) Visualizzazione dei segnali in bassa risoluzione in attesa della risposta dal server



(c) Visualizzazione dei segnali alla risoluzione corretta

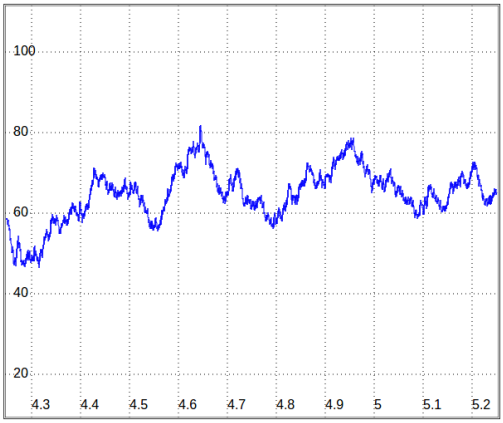
Figura 5.1: Fasi in cui si svolge l'operazione di zoom

5.1.3 Gestione del pan

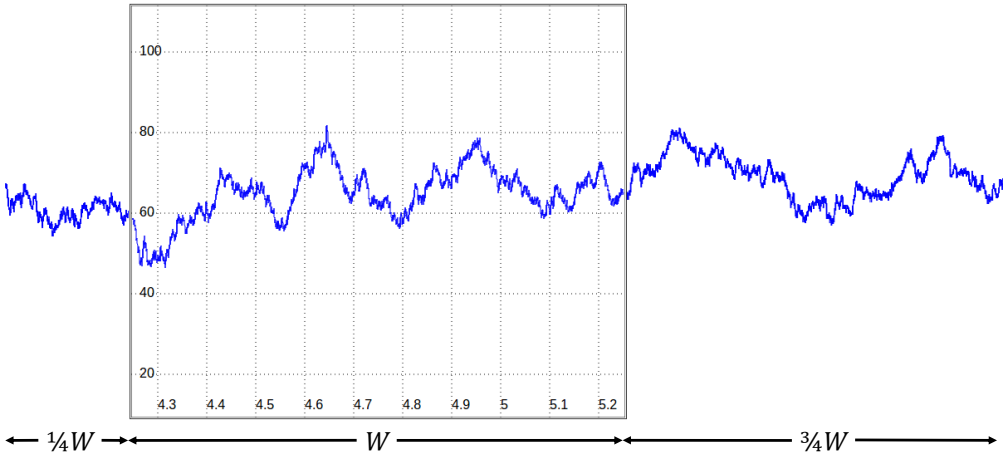
In seguito ad un'operazione di zoom solamente la porzione di segnale ingrandita è disponibile alla risoluzione corretta, per cui una successiva operazione di pan richiederà anch'essa un aggiornamento dei dati dei segnali. Durante il pan, mentre l'utente sposta il mouse l'area visualizzata cambia continuamente, pertanto una gestione uguale a quella dello zoom non sarebbe soddisfacente, perché verrebbero continuamente inviate richieste al server per intervalli dei segnali leggermente diversi gli uni dagli altri, e nel momento in cui verrebbe ricevuta una risposta l'area da visualizzare sarebbe probabilmente cambiata.

La soluzione adottata consiste nel richiedere, in seguito ad un'operazione di pan, un intervallo di ascisse di ampiezza doppia rispetto a quello visualizzato, con un numero di punti pari al doppio del valore di default ($2N_{def}$) in modo da mantenere la risoluzione; così facendo, finché l'area visualizzata rimane all'interno dell'intervallo non è necessario eseguire aggiornamenti. Inoltre dato che, se l'utente sta spostando l'area visualizzata in una direzione (verso destra o verso sinistra), è probabile che continui a farlo nella stessa direzione, l'intervallo richiesto non è centrato su quello visualizzato al momento, ma si estende maggiormente nella direzione dello spostamento: indicando con $[x_0, x_1]$ l'intervallo visualizzato subito dopo un'operazione di zoom e con $W = x_1 - x_0$ la sua ampiezza, nel momento in cui viene eseguito un pan verso destra verrà richiesto l'intervallo $[x_0 - 1/4W, x_1 + 3/4W]$, cioè un intervallo di ampiezza $2W$ che si estende per $3/4W$ verso destra e $1/4W$ verso sinistra (figura 5.2b). Viceversa se il pan è effettuato verso sinistra verrà richiesto l'intervallo $[x_0 - 3/4W, x_1 + 1/4W]$.

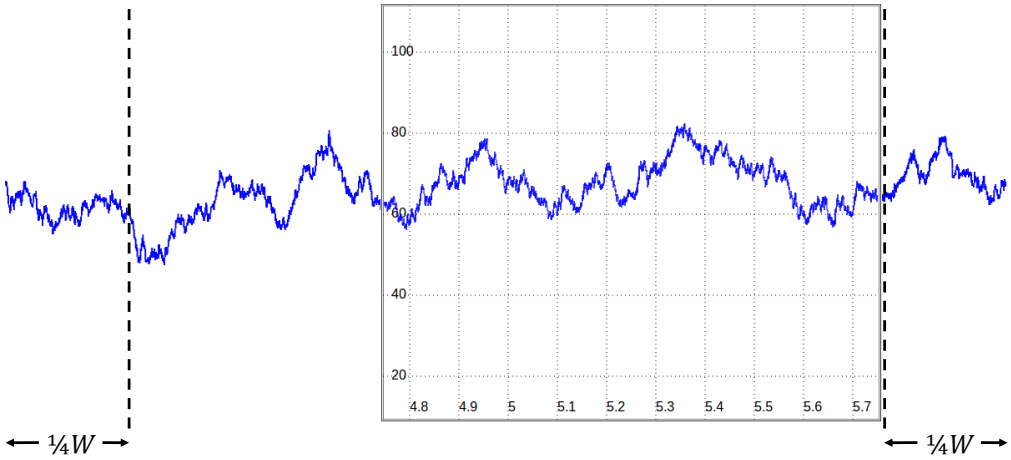
In seguito a successive operazioni di pan, quando l'area visualizzata esce dall'intervallo in cui il client ha a disposizione i dati dei segnali alla giusta risoluzione, dovrà essere richiesto al server un nuovo intervallo, e in attesa della risposta, come nel caso dello zoom, verranno visualizzati i segnali in bassa risoluzione, che vengono sempre mantenuti in memoria per essere usati come fallback quando necessario (in altre parole sono sempre presenti, per ogni pannello, due array di oggetti `Signal`: uno con i segnali alla giusta risoluzione nell'intervallo corrente e uno con i segnali ottenuti in seguito alla richiesta effettuata nel momento della creazione del pannello, quindi in tutto l'intervallo in cui sono definiti ma a bassa risoluzione). Per evitare il più possibile di visualizzare i segnali in bassa risoluzione, la richiesta di aggiornamento viene eseguita prima che l'area visualizzata esca dall'intervallo corrente, in particolare quando uno dei due estremi di ascisse dell'area visualizzata arriva ad essere distante meno di $1/4W$ da un estremo dell'intervallo corrente (figura 5.2c). In questo modo è probabile che, quando l'utente avrà spostato l'area visualizzata fuori dall'intervallo precedentemente usato, sarà già stata ricevuta la risposta dal server con i dati dei segnali nel nuovo intervallo. La scelta di usare i valori descritti, $2W$ per l'ampiezza dell'intervallo, $1/4W$ e $3/4W$ per gli offset dell'intervallo richiesto e $1/4W$ per la soglia oltre la quale viene effettuato un aggiornamento, è stata fatta dopo alcune prove in modo da minimizzare il numero di punti richiesti ad ogni aggiornamento e la frequenza degli aggiornamenti mantenendo al tempo stesso una buona *user experience*.



(a) Segnale visualizzato subito dopo un'operazione di zoom



(b) Segnale che viene richiesto al server dopo un'operazione di pan verso destra



(c) Un nuovo aggiornamento viene effettuato quando l'area visualizzata supera una delle due soglie, mostrate in figura con una linea tratteggiata

Figura 5.2: Gestione del pan

5.2 Visualizzazione di sequenze di frame

Oltre a semplici segnali, MDSplus permette di memorizzare anche sequenze di frame acquisite mediante telecamere. Ciascun frame è rappresentato come un segnale avente due dimensioni, una per la larghezza e una per l'altezza del frame, con l'elemento in posizione (i, j) costituito da un valore intero a 8, 16 o 32 bit che rappresenta l'intensità del pixel di coordinate (i, j) nel frame (le immagini acquisite sono in scala di grigi, pertanto un pixel contiene solo l'informazione sull'intensità); ad ogni frame è poi associato un istante temporale che può essere relativo all'inizio della sequenza o assoluto, nel qual caso esso è un timestamp il cui valore indica il numero di millisecondi trascorsi dal 1 gennaio 1970 (lo Unix Epoch).

Per motivi di prestazioni i frame sono memorizzati in segmenti (vedi paragrafo 2.2.3), in modo che sia possibile accedere ad un particolare frame senza leggere l'intera sequenza, che nel suo insieme può assumere dimensioni notevoli (ad esempio, una sequenza di 50 frame a 16 bit con una risoluzione di 1920×1080 pixel richiede oltre 200 megabyte). Anche se è possibile memorizzare più di un frame in un unico segmento, comunemente si usa un segmento separato per ciascun frame, impostando i valori del tempo di inizio e del tempo di fine del segmento all'istante temporale associato al frame.

Nei prossimi paragrafi vengono esposte le modifiche apportate a WebScope per introdurre la visualizzazione delle sequenze di frame e le operazioni ad esse associate, ossia la riproduzione di una sequenza e lo zoom, il pan e lo scaling di un frame.

5.2.1 Il tipo di richiesta `frame`

Date le differenze tra i segnali semplici e quelli che rappresentano sequenze di frame, a lato server è stata implementata la gestione di un nuovo tipo di richiesta, che permette di ottenere i dati relativi ad un frame e l'array degli istanti temporali associati ai frame di una sequenza. Essa ha la forma `/frame?<opzione1>=<espressione1>&<opzione2>=<espressione2>&...`, dove le opzioni possibili sono `title`, `xlabel`, `ylabel`, `tree`, `shot`, `frame_idx`, `init`, `y` e `x`.

Le prime cinque hanno lo stesso significato che assumono nel caso del tipo di richiesta `scope`, ossia definiscono le espressioni del titolo del pannello, delle etichette degli assi, del *tree* e dello *shot* in cui è memorizzato il segnale che rappresenta la sequenza di frame; `frame_idx` permette di specificare l'indice, all'interno della sequenza, del frame che si vuole ricevere, e se non è presente viene assunto il valore di default 0, cioè l'indice del primo frame; `init` indica se la richiesta è quella che viene eseguita al momento della creazione del pannello (o dopo un cambio di *shot* tramite il campo "shots" dell'applicazione) o se invece è effettuata successivamente: la differenza sta nel fatto che l'array degli istanti temporali viene incluso nella risposta solo alla prima richiesta per un dato pannello; infine le opzioni `y` e `x` specificano rispettivamente l'espressione del segnale che rappresenta la sequenza di frame e quella dell'eventuale segnale da usare come array degli istanti temporali, al posto dei valori ricavati dai tempi di inizio e fine dei segmenti in cui è memorizzata la sequenza.

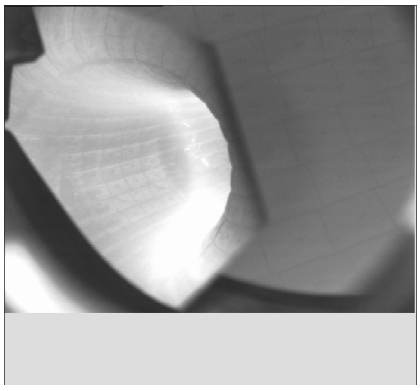
In seguito alla valutazione delle espressioni tramite l'interfaccia a oggetti di MDSplus, viene inviata come risposta la sequenza di byte che corrisponde ai dati del frame, ossia uno dei tipi di array MDSplus con valori interi, e, se l'opzione `init` è presente, l'array degli istanti temporali. Le informazioni incluse negli header HTTP comprendono la lunghezza dell'array con i dati del frame ed eventualmente di quello con gli istanti temporali, la larghezza e l'altezza del frame (ricavate dalle lunghezze delle due dimensioni del segnale che rappresenta la sequenza di frame) e il numero di byte usati per rappresentare un pixel del frame (1, 2 o 4).

5.2.2 Opzioni di configurazione relative ai frame

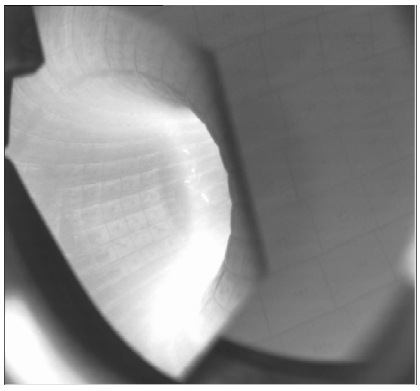
Una sequenza di frame viene visualizzata all'interno di un pannello che può presentare nel file di configurazione, oltre a quelle descritte nel paragrafo 4.2.1, altre opzioni specifiche per i frame:

- `is_image`: indica che il pannello contiene una sequenza di frame e non semplici segnali.
- `keep_ratio`: indica se mantenere l'*aspect ratio* dei frame nella visualizzazione o se modificarlo per riempire tutto il pannello.
- `horizontal_flip` e `vertical_flip`: indicano se capovolgere orizzontalmente e/o verticalmente i frame.
- `palette`: indica il nome di una palette colori da usare per colorare i frame, che, come detto in precedenza, sono immagini in scala di grigi.
- `bit_shift` e `bit_clip`: permettono di modificare il modo in cui la palette colori viene applicata, come spiegato nel seguito del paragrafo. `bit_shift` assume un valore intero positivo o negativo, `bit_clip` un valore booleano.

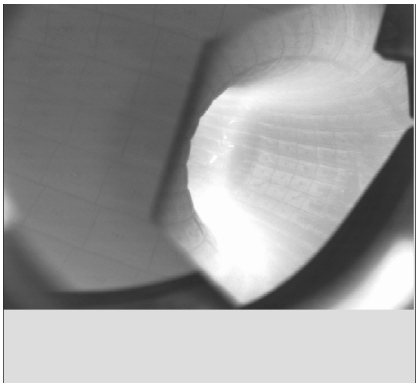
In figura 5.3 viene mostrato un esempio dei risultati ottenuti applicando ad uno stesso frame le diverse opzioni di configurazione.



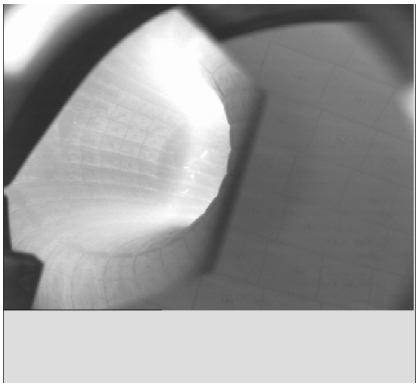
(a) Frame con opzioni di default



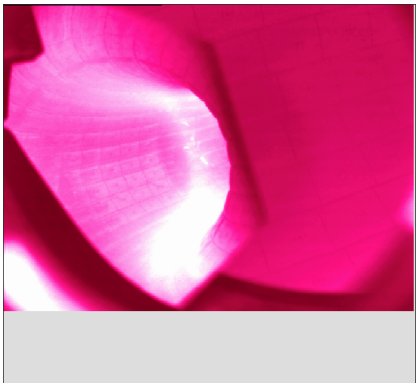
(b) Frame con opzione `keep_ratio` impostata a `false`



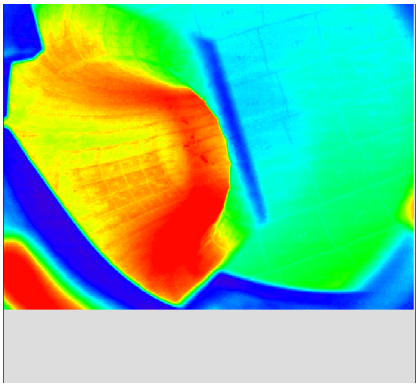
(c) Frame con opzione `horizontal_flip` impostata a `true`



(d) Frame con opzione `vertical_flip` impostata a `true`



(e) Frame con opzione `palette` impostata a "RED-PURPLE"



(f) Frame con opzione `palette` impostata a "RAINBOW"

Figura 5.3: Opzioni di configurazione relative ai frame

È stata quindi modificata a lato server la funzione di inizializzazione che gestisce il tipo di richiesta `scope` nella variante `configxml`, in modo da includere queste opzioni nella conversione del file di configurazione in formato XML, e a lato client la funzione che si occupa del *parsing* del file.

Le palette colori sono memorizzate in un file in formato binario che contiene, per ogni palette, una stringa che ne rappresenta il nome e tre array di byte con 256 elementi ciascuno, che definiscono la funzione di conversione dall'intensità di un pixel del frame ai tre valori R, G e B del corrispondente pixel nella versione colorata del frame, che è quella che poi viene visualizzata nel pannello. Ad esempio, dato un pixel nel frame di intensità 50 e dati i tre array `r`, `g` e `b` associati ad una certa palette, il colore del pixel da visualizzare nel pannello avrà come componenti RGB `r[50]`, `g[50]` e `b[50]`. Poiché i tre array `r`, `g` e `b` di ogni palette hanno solo 256 elementi, un indice al loro interno non può assumere valori al di fuori dell'intervallo $[0, 255]$, pertanto nel caso di frame a 16 o 32 bit i pixel con intensità maggiori di 255 non vengono visualizzati (o meglio, vengono

visualizzati come pixel neri). Questo comportamento può essere modificato con le opzioni `bit_shift` e `bit_clip`: la prima permette di applicare al valore dell'intensità dei pixel uno shift del numero di posizioni specificato, verso sinistra se il valore dell'opzione è positivo e verso destra se negativo; la seconda, se impostata a `true`, fa sì che i pixel che hanno intensità (dopo aver applicato l'eventuale bit shift) maggiore di 255, vengano visualizzati come se avessero intensità massima (255) anziché come pixel neri (intensità 0).

La figura 5.4 mostra un esempio degli effetti delle opzioni `bit_shift` e `bit_clip` su un frame a 16 bit creato via software, costituito da 1000 linee la cui intensità cresce da 0 (linea più in alto) a 999 (linea più in basso), al quale è stata applicata la palette di default "B-W LINEAR" che associa ad intensità maggiori pixel più chiari su una scala di grigi. Considerando ad esempio la linea di intensità 500, nella prima figura essa viene visualizzata in nero, perché l'intensità è maggiore di 255, mentre nella seconda e nella terza viene visualizzata come se avesse intensità $500 >> 1 = 250$, quindi con un grigio molto chiaro; nella terza figura l'opzione `bit_clip` è impostata a `true` e quindi tutte le linee di intensità I tale che $(I >> 1) > 255$ vengono mostrate come se avessero intensità 255, quindi in bianco.

(a) `bit_shift`=0, `bit_clip`=`false`

(b) `bit_shift`=−1, `bit_clip`=`false`

(c) `bit_shift`=−1, `bit_clip`=`true`

Figura 5.4: Effetti delle opzioni `bit_shift` e `bit_clip`

Per fare in modo che il client possa ottenere il file con le palette colori è stata introdotta la nuova variante `colortables` del tipo di richiesta `scope`, ed è stata aggiunta tra le funzioni di inizializzazione del client una che esegue la richiesta e il *parsing* del file per costruire una struttura dati che mappa il nome di ciascuna palette nell'insieme dei corrispondenti array `r`, `g` e `b`, la quale verrà poi usata durante la procedura di visualizzazione dei frame.

5.2.3 Pannelli per la visualizzazione dei frame e la classe `FramePanel`

I frame vengono visualizzati in WebScope all'interno di pannelli che possono essere inseriti nell'area di visualizzazione insieme a quelli che contengono segnali semplici, come mostrato in figura 5.5. Le operazioni di zoom, pan, crosshair, scaling e ridimensionamen-

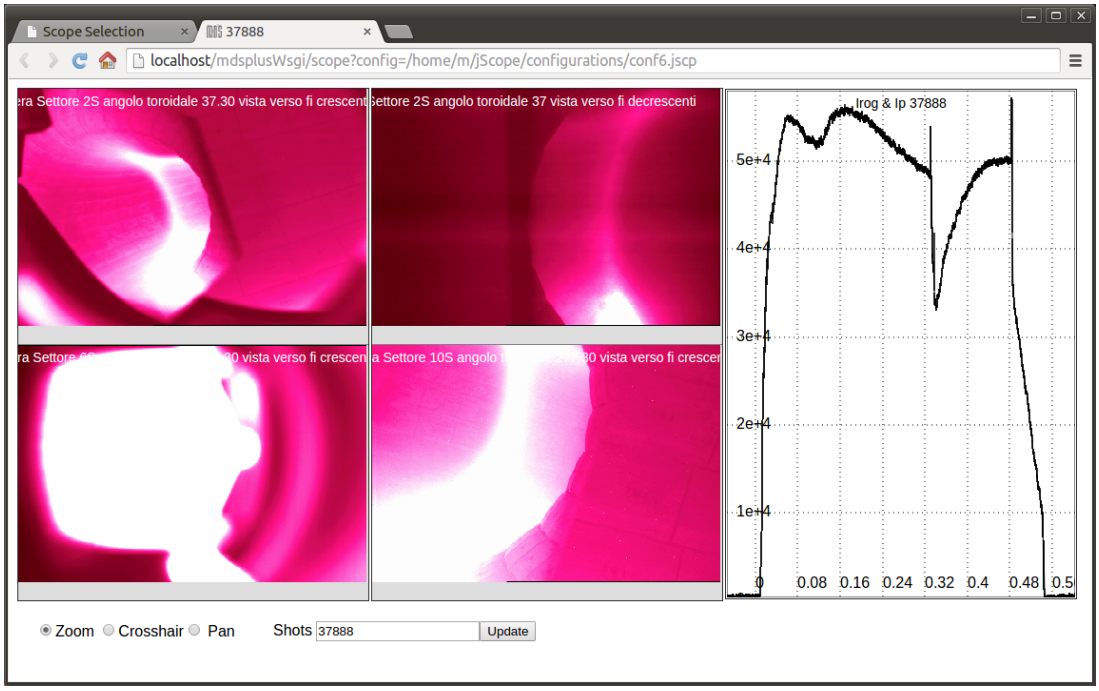


Figura 5.5: Esempio di una configurazione contenente pannelli per la visualizzazione dei frame

to del pannello sono supportate anche per i frame, con la differenza che le uniche possibilità di scaling, accessibili nel menù che compare facendo click destro su un pannello, sono Autoscale e Autoscale All Images, che permettono di ripristinare la visualizzazione dell'intero frame rispettivamente nel pannello corrente e in tutti i pannelli contenenti frame.

Mentre i pannelli in cui vengono visualizzati i segnali semplici sono costituiti ciascuno da un elemento `<svg>`, per i pannelli contenenti frame vengono usati un elemento `<canvas>`, in cui viene disegnato il frame, e, sovrapposto ad esso, un elemento `<svg>`, in cui vengono inseriti gli elementi grafici corrispondenti al titolo del pannello, il rettangolo visualizzato durante lo zoom e le due linee visualizzate durante il crosshair. L'elemento `<svg>` viene utilizzato in quanto, pur essendo possibile ottenere gli stessi risultati usando solo l'elemento `<canvas>`, il procedimento sarebbe meno efficiente, dato che tutti gli elementi grafici in `<canvas>` sono disegnati come bitmap, e quindi non sarebbe possibile ad esempio ridimensionare solo il rettangolo per lo zoom quando l'utente sposta il mouse, ma bisognerebbe ridisegnare l'intero pannello, compreso il frame.

All'avvio dell'applicazione, per ogni pannello contenente frame, una delle funzioni di inizializzazione del client crea i due elementi `<svg>` e `<canvas>` e un'istanza della classe `FramePanel`, che viene introdotta per gestire i pannelli di questo tipo. La struttura della classe è in parte simile a quella di `WavePanel`, con una funzione per l'aggiornamento dei dati del pannello (`update`), per il ridimensionamento dello stesso (`resize`), per l'ope-

razione di autoscale (`autoscale`) e per la gestione di zoom, pan e crosshair, ciascuna effettuata da tre funzioni come ad esempio `startZoom`, `dragZoom` e `endZoom`. Sono poi presenti altre funzioni specifiche per i pannelli contenenti frame, ossia quelle per la riproduzione della sequenza di frame (`startPlay` e `stopPlay`) e per la visualizzazione di un frame (`drawFrame`).

5.2.4 Visualizzazione di un frame

Il procedimento che porta alla visualizzazione di un frame inizia con l'esecuzione di una richiesta di tipo `frame` al server. I dati ricevuti vengono inseriti all'interno di un oggetto `ImageData` (vedi sezione 3.3) applicando prima a ciascun pixel del frame le opzioni `palette`, `bit_shift` e `bit_clip`; l'oggetto `ImageData` viene quindi usato per aggiornare la bitmap dell'elemento `<canvas>`, che viene poi ridimensionato in base a quanto stabilito dal valore dell'opzione `keep_ratio`, e infine vengono applicate le opzioni `horizontal_flip` e `vertical_flip` agendo sull'oggetto `Context` associato all'elemento `<canvas>`.

Per consentire l'implementazione delle funzionalità di zoom, pan, autoscale e ridimensionamento del pannello viene utilizzato un secondo elemento `<canvas>`, che non viene visualizzato, e che ha le stesse dimensioni del frame (al contrario dell'elemento `<canvas>` primario, le cui dimensioni dipendono da quelle del pannello che lo contiene). Questo secondo `<canvas>` viene mantenuto aggiornato, come il primario, ogni volta che è richiesta la visualizzazione di un nuovo frame.

Quando l'utente effettua un'operazione di zoom o pan, viene individuato sul `<canvas>` secondario il rettangolo corrispondente alla porzione del frame da visualizzare, che viene copiata sul `<canvas>` primario con la funzione `drawImage` dell'oggetto `Context` ad esso associato. Allo stesso modo le operazioni di autoscale e ridimensionamento sono effettuate ridisegnando il frame sul `<canvas>` primario usando come sorgente il secondario. Quest'ultimo rimane immutato finché il frame visualizzato non cambia, e rappresenta quindi il frame così come è stato ricevuto dal server, mentre il primario viene continuamente modificato dalle operazioni effettuate dall'utente.

5.2.5 Riproduzione di sequenze di frame

Per riprodurre una sequenza di frame vengono resi disponibili i pulsanti "Start play" e "Stop play" nel menù che compare facendo click destro su un pannello. La sequenza viene riprodotta a partire dal frame corrente, visualizzando i frame successivi (ottenuti eseguendo richieste di tipo `frame` al server) ad intervalli di un secondo l'uno dall'altro; inoltre, se la modalità selezionata è "crosshair", la riproduzione avviene in maniera sincronizzata con gli altri pannelli, ossia,

- in ogni altro pannello contenente frame vengono visualizzati i frame delle rispettive sequenze negli stessi istanti, se esistono, altrimenti un messaggio informa l'utente della loro assenza.

- in ogni altro pannello contenente segnali semplici viene mostrato un crosshair che si sposta in corrispondenza delle ascisse pari ai valori degli istanti temporali.

Ad esempio, supponendo di avere una configurazione con un pannello contenente la sequenza di frame A negli istanti 0, 0.1, 0.2 e 0.3, uno con un'altra sequenza B negli istanti 0, 0.1 e 0.2, e uno con dei segnali nell'intervallo di ascisse $[0, 0.3]$, inizialmente saranno visualizzati i frame delle due sequenze nell'istante 0; facendo click destro sul primo pannello ed eseguendo "Start play" verranno visualizzati, dopo un secondo, i frame delle sequenze A e B all'istante 0.1 nei rispettivi pannelli, e un crosshair in corrispondenza dell'ascissa 0.1 nel terzo pannello. Dopo tre secondi, nel primo pannello sarà mostrato il frame all'istante 0.3, nel secondo un messaggio che indica l'assenza di un frame all'istante 0.3 nella sequenza B, e nel terzo il crosshair nell'ascissa 0.3. Terminata la sequenza nel pannello su cui è stato eseguito "Start play", viene di nuovo visualizzato il primo frame e il ciclo ricomincia, fino a che la riproduzione non viene fermata con "Stop play".

Questo tipo di visualizzazione sincronizzata è utile ad esempio quando le diverse sequenze di frame mostrano uno stesso fenomeno ripreso da molteplici angolazioni, e i segnali rappresentano il variare di alcune grandezze fisiche negli stessi istanti.

In alternativa all'uso dei pulsanti "Start play" e "Stop play", quando si è in modalità crosshair ed è presente almeno un pannello contenente segnali semplici, è possibile visualizzare i diversi frame di una sequenza spostando il crosshair su di esso: in tutti i pannelli contenenti frame verranno visualizzati i frame delle rispettive sequenze nell'istante dato dall'ascissa su cui si trova il crosshair.

5.3 Visualizzazione di segnali in tempo reale

Un'applicazione come WebScope può essere utile, oltre che per visualizzare i dati relativi ad esprimenti già conclusi, anche per monitorare lo svolgimento di esprimenti in corso, visualizzando i nuovi dati acquisiti in tempo reale.

La visualizzazione in tempo reale viene realizzata facendo in modo che il client effettui richieste al server, a intervalli regolari (per impostazione predefinita, distanziate di un secondo l'una dall'altra), per verificare la disponibilità di nuovi dati, che, se presenti, vengono inviati nella risposta. Per indicare che i dati di un pannello devono essere visualizzati in tempo reale viene usata l'opzione `continuous_update`, la cui introduzione in WebScope ha richiesto l'aggiornamento delle funzioni per la gestione dei file di configurazione, sia lato server che lato client, come nel caso delle opzioni relative ai frame. Quando un pannello ha l'opzione `continuous_update` impostata a `true`, indicando con $[x_0, x_1]$ l'intervallo di ascisse su cui sono definiti i segnali in un certo istante, a seconda della porzione di segnali visualizzata si possono avere diversi comportamenti:

- se sono visualizzati i segnali nella loro interezza, ossia nell'intervallo $[x_0, x_1]$, nel momento in cui vengono resi disponibili nuovi dati l'intervallo visualizzato aumenta, in modo da mostrare ancora i segnali completi; esso diventa quindi $[x_0, x_1 + \Delta x]$, dove Δx è l'ampiezza dell'intervallo costituito dai nuovi dati. In questo modo si

ha che, progressivamente, i segnali che erano visualizzati inizialmente vengono "compressi" verso sinistra per far posto ai nuovi dati.

- se l'intervallo visualizzato è del tipo $[x_i, x_j]$ con $x_i > x_0$ e $x_j < x_1$, ossia la visualizzazione è ingrandita su una certa porzione dei segnali e l'area visualizzata non include l'estremo destro x_1, essa non viene modificata anche se sono disponibili nuovi dati.

- se l'intervallo visualizzato è del tipo $[x_i, x_j]$ con $x_i > x_0$ e $x_j \geq x_1$, ossia la visualizzazione è ingrandita e l'area visualizzata comprende l'estremo destro x_1, quando vengono resi disponibili nuovi dati essa viene spostata verso destra di un intervallo di ampiezza Δx (diventando cioè $[x_i + \Delta x, x_j + \Delta x]$), in modo da visualizzare i nuovi dati e parte di quelli precedenti, come se fosse stato eseguito un pan verso destra. Con il continuo arrivo di nuovi dati i segnali "scorrono" quindi da destra verso sinistra.

A lato client ogni oggetto `WavePanel` corrispondente ad un pannello con l'opzione `continuous_update` impostata a `true` controlla, ad intervalli di un secondo, gli estremi di ascisse dell'area visualizzata al momento e, se la situazione è quella descritta dal primo o dal terzo caso, invia una richiesta al server per verificare la presenza di nuovi dati. Quest'ultima è una richiesta di tipo `scope` nella variante `panel`, con l'aggiunta di nuove opzioni: `continuous_update`, la cui presenza serve a differenziare la richiesta da quelle "standard" effettuate ad esempio in seguito ad un'operazione di zoom o pan, `displayed_width`, che indica l'ampiezza dell'intervallo di ascisse attualmente visualizzato, `default_num_samples`, corrispondente al valore N_{def} introdotto nel paragrafo 5.1.1, che è un dato necessario, insieme a `displayed_width`, per il calcolo del numero di punti da usare nel resampling degli eventuali nuovi dati, e infine N opzioni `curr_max_xi` che indicano il massimo valore corrente della componente x di ciascun segnale i del pannello.

Il server, ricevendo una richiesta di questo tipo, ricava per ogni segnale il massimo valore della sua componente x, in seguito chiamato `new_max_xi`, e, se esso è maggiore del valore indicato da `curr_max_xi`, procede alla valutazione di un'espressione contenente la funzione `GetXYWave` e i relativi parametri, dati dall'espressione del segnale, dall'intervallo di ascisse richiesto e dal numero di punti richiesti.

L'espressione del segnale è contenuta, per il segnale i-esimo, nell'opzione `yi` della richiesta. Come estremo sinistro dell'intervallo viene usato il valore `curr_max_xi + epsilon`, dove `epsilon` è un valore molto piccolo[1], se la componente x del segnale ha un tipo di dato *floating point*, altrimenti viene usato l'intero successivo a `curr_max_xi`; in entrambi i casi il valore viene scelto per fare in modo che il punto di ascissa `curr_max_xi`, già presente tra i dati del client, non sia incluso tra i nuovi dati. Come estremo destro dell'intervallo viene usato il valore `new_max_xi`, mentre il numero di punti richiesti è dato dalla quantità `(new_max_xi-curr_max_xi)*default_num_samples/displayed_width`,

[1] In particolare esso è uguale al più piccolo valore positivo rappresentabile con il tipo di dato usato per la componente x del segnale.

ossia l'ampiezza dell'intervallo dei nuovi dati moltiplicata per il numero di punti per unità di larghezza usati dal pannello.

I nuovi dati vengono quindi inviati nella risposta al client, che li usa per aggiornare gli oggetti `Signal` del pannello che ha effettuato la richiesta e modifica l'area visualizzata come descritto in precedenza.

5.4 Altre funzionalità

5.4.1 Undo zoom

Durante l'utilizzo dell'applicazione capita spesso che, dopo aver effettuato più zoom consecutivi su una certa regione di un pannello, l'utente desideri ritornare al livello di zoom immediatamente precedente. È stata quindi introdotta questa possibilità fra le opzioni disponibili nel menù che compare facendo clic destro su un pannello. Per ottenere il comportamento richiesto viene mantenuto a lato client, all'interno della classe `WavePanel`, uno stack di oggetti `Wave` che corrispondono ciascuno all'insieme dei segnali visualizzati ad un livello di zoom: ogni volta che viene eseguita un'operazione di zoom, l'oggetto `Wave` corrente viene aggiunto in cima allo stack, mentre all'esecuzione di un undo zoom un oggetto `Wave` viene rimosso dalla cima dello stack e usato per la visualizzazione dei segnali; questa implementazione permette di effettuare l'undo zoom senza eseguire richieste al server, perché tutti i dati necessari sono presenti nello stack. Nel momento in cui viene poi fatta un'operazione di pan, autoscale o reset scales, lo stack viene svuotato.

5.4.2 Gestione dei tempi assoluti

Come accennato nella sezione 5.2, ad un frame può essere associato un istante temporale assoluto, ossia un timestamp. Allo stesso modo, un segnale semplice può avere come componente x un array di timestamp, ciascuno dei quali indica l'istante in cui il corrispondente valore della componente y è stato acquisito. Nella visualizzazione di un segnale che presenta questa caratteristica, sarebbe di scarsa utilità mostrare i valori numerici dei timestamp come indicatori lungo l'asse x del pannello, perché essi rappresentano in realtà date e orari: per questo motivo è stata modificata la procedura di creazione delle griglie dei pannelli, in modo da gestire questo tipo particolare di segnale.

I timestamp sono valori interi a 64 bit, quindi se la componente x di un segnale è costituita da un array di timestamp essa avrà come tipo di dato Int64Array o Uint64Array; in pratica, questi due tipi di dato sono usati per la componente x di un segnale solo se essa rappresenta un array di timestamp, quindi è possibile stabilire se un segnale richiede una gestione particolare dei valori della componente x semplicemente controllando il tipo di dato di quest'ultima.

La creazione degli indicatori lungo l'asse x del pannello viene modificata visualizzando i valori dei timestamp convertiti in date e orari, e in particolare, a seconda dell'intervallo di ascisse visualizzato nel pannello e dei valori dei timestamp al suo interno, la visualizzazione viene effettuata in questo modo:

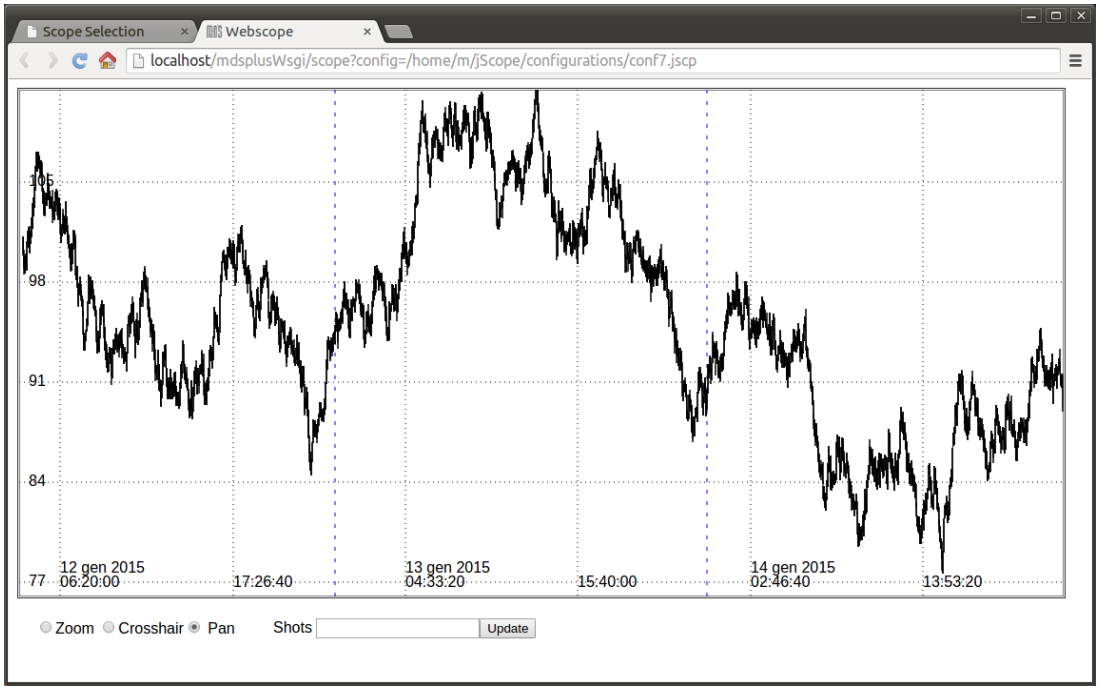


Figura 5.6: Esempio di visualizzazione di un segnale la cui componente x rappresenta una sequenza di tempi assoluti.

- se i timestamp di tutti gli indicatori nell'intervallo fanno parte della stessa data, vengono mostrati solo gli orari;
- se un sottoinsieme degli indicatori ha i timestamp appartenenti alla stessa data, essa viene mostrata solo sul primo indicatore del sottoinsieme;
- se l'intervallo di tempo tra gli orari visualizzati in due indicatori consecutivi è inferiore a un secondo, oltre all'orario vengono visualizzati anche i millisecondi;
- se l'intervallo comprende più date, vengono visualizzate delle linee di separazione tra una data e l'altra, di colore blu e tratteggiate.

Nell'esempio in figura 5.6, coppie di indicatori consecutivi hanno timestamp che appartengono alla stessa data, che viene quindi mostrata solo sul primo dei due, e inoltre, dato che l'intervallo visualizzato comprende tre giorni, sono mostrate due linee di separazione delle date.

Capitolo 6

Conclusioni

L'obiettivo dell'attività di tesi era la realizzazione di nuove funzionalità per l'applicazione di visualizzazione di dati WebScope. La nuova versione dell'applicazione è ora in grado di gestire la visualizzazione di segnali con un numero particolarmente alto di punti, configurazioni che contengono sequenze di frame e dati acquisiti in tempo reale. Le funzionalità implementate sono state testate utilizzando i dati dell'esperimento RFX in corso presso l'Istituto Gas Ionizzati e i relativi file di configurazione, individuando e correggendo alcune problematiche riscontrate. Come accade spesso nello sviluppo di applicazioni web, è stato necessario verificare il corretto funzionamento di WebScope con i principali browser e gestire alcune differenze che essi presentano nell'implementazione delle tecnologie utilizzate dall'applicazione, in particolare alcuni aspetti del linguaggio JavaScript e della gestione dell'elemento canvas.

Per quanto riguarda le possibili funzionalità da aggiungere in futuro, sarebbe utile la capacità di utilizzare file di configurazione presenti sul client, dato che al momento possono essere selezionati solo quelli memorizzati sul server. Inoltre, anche se i file di configurazione vengono modificati raramente, potrebbe risultare conveniente mettere a disposizione un'interfaccia grafica che consenta di cambiare le varie impostazioni di configurazione di ciascun pannello, aggiungere o rimuovere pannelli e memorizzare la nuova configurazione in un file, come è possibile fare in jScope. Infine potrebbero essere implementate funzionalità di visualizzazione avanzate come la rappresentazione di dati in tre dimensioni.

Bibliografia

[1] G. Manduchi, T. Fredian, and J. Stillerman. A new web-based tool for data visualization in MDSplus. *Fusion Engineering and Design*, 89(5), 2014.

[2] G. Manduchi, T. Fredian, and J. Stillerman. A new object-oriented interface to MDSplus. *Fusion Engineering and Design*, 85(3–4), 2010.

[3] Istituto Gas Ionizzati
`http://www.igi.cnr.it/`

[4] Introduction - MdsWiki
`http://www.mdsplus.org/index.php?title=Introduction`

[5] Documentation - MdsWiki
`http://www.mdsplus.org/index.php?title=Documentation`

[6] Introduction to MDSplus
`http://www.mdsplus.org/old/ga_jschachter/mdsplus-class1.pdf`
`http://www.mdsplus.org/old/ga_jschachter/mdsplus-class2.pdf`

[7] JavaScript introduction
`http://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction`

[8] AJAX documentation
`http://developer.mozilla.org/en/docs/AJAX`

[9] Ajax: A New Approach to Web Applications
`http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/`

[10] An SVG Primer for Today's Browsers
`http://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html`

[11] SVG documentation
`http://developer.mozilla.org/en/docs/Web/SVG`

[12] Scalable Vector Graphics
`http://en.wikipedia.org/wiki/Scalable_Vector_Graphics`

[13] The canvas element
http://www.w3.org/TR/html5/scripting-1.html#the-canvas-element

[14] Canvas documentation
http://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

[15] Python
http://en.wikipedia.org/wiki/Python_(programming_language)

[16] PEP 333 - Python Web Server Gateway Interface v1.0
http://www.python.org/dev/peps/pep-0333/

[17] An Introduction to the Python Web Server Gateway Interface (WSGI)
http://ivory.idyll.org/articles/wsgi-intro/what-is-wsgi.html

[18] WSGI tutorial
http://webpython.codepoint.net/wsgi_tutorial

[19] mod_wsgi - Python WSGI adapter module for Apache
http://code.google.com/p/modwsgi/