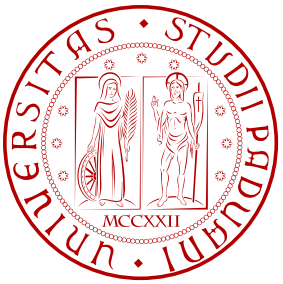


Università degli Studi di Padova

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in Ingegneria Elettronica

TESI DI LAUREA TRIENNALE

Controllo remoto di un multimetro digitale: sviluppo di web-server in LabVIEW e client JavaFX

Relatore:
Dr. *Giada Giorgi*

Candidato:
Riccardo Ciatto

Anno accademico 2010/2011

INDICE

CAPITOLO

1

INTRODUZIONE

Le potenzialità della formazione a distanza, ed in particolare dell'uso di Internet quale strumento per fornire in maniera veloce materiale formativo, sono ormai riconosciute da tempo. Nel campo delle metodologie didattiche, ci si è orientati verso un sempre maggiore utilizzo di sistemi di apprendimento interattivi, e possibilmente, anche utilizzabili a distanza (o come si dice comunemente, "in modalità remota").

Grazie alla capillare diffusione della rete, alla standardizzazione dei protocolli e alla possibilità di sviluppare facilmente interfacce grafiche di utente (Graphical User Interfaces), il web si rivela un formidabile ambiente per lo sviluppo di applicazioni didattiche avanzate.

Il controllo remoto della strumentazione e l'esecuzione di esperimenti reali via Internet sono da alcuni anni argomenti di interesse per molti ricercatori operanti in differenti campi. Quello che si vuole realizzare è un vero e proprio laboratorio utilizzabile a distanza (da casa), ove l'effetto "presenza" all'interno del laboratorio viene ottenuto grazie allo sviluppo di particolari interfacce grafiche di utente.

Un laboratorio remoto è un sistema hardware/software che consente agli utenti di interagire con processi fisici dislocati in altri luoghi attraverso la rete Internet (od altri tipi di reti). Le possibili applicazioni di questi laboratori sono molteplici e spaziano dalla programmazione a distanza di robot,

macchinari utensili, strumentazione, all'utilizzo di tali strumenti nell'ambito didattico al fine di facilitare lo svolgimento di esercitazioni pratiche.

In ambito didattico, la particolarità, consiste nell'offrire all'utenza la possibilità di eseguire da remoto, esperimenti su strumentazione reale in qualsiasi momento della giornata, da qualsiasi luogo, per il numero di volte che si ritiene necessario. L'utilizzo di Internet consente una gestione più efficiente delle risorse di laboratorio che diventano fruibili durante tutto l'arco temporale della giornata e da qualunque computer collegato alla rete.

In merito a quanto detto, supponiamo che il Dipartimento di Ingegneria dell'Informazione dell'Università di Padova disponga di un laboratorio remoto di misure elettroniche. Le motivazioni che possono aver spinto i docenti alla realizzazione di un laboratorio di misure sono di seguito riassunte:

- Maggiore possibilità di approfondire le tematiche della misurazione elettronica: attraverso questo strumento gli studenti sono maggiormente motivati ad effettuare esperienze pratiche di misurazione, acquisendo una più completa conoscenza della materia.

- Possibilità di lavorare su dati reali: il sistema consente allo studente di lavorare su dati reali, effettivamente acquisiti in laboratorio dallo strumento hardware, e non su di un convenzionale simulatore software.

- Didattica remota: mediante l'uso di un laboratorio remoto è possibile effettuare lezioni o corsi che utilizzano dei processi fisici pur non trovandosi fisicamente nei pressi degli stessi.

- Piena accessibilità: a differenza dei laboratori tradizionali, un laboratorio remoto può essere utilizzato 24 ore al giorno e da qualunque terminale collegato ad Internet. Questa caratteristica può essere sfruttata con vantaggio dagli studenti che possono quindi effettuare esperienze ed esercizi pratici anche da casa ed in qualunque orario.

- Più facile utilizzazione delle attrezzature: il sistema offre allo studente la possibilità di utilizzare anche attrezzature e strumentazione particolarmente sofisticate e costose, che altrimenti sarebbe difficile rendere disponibile in copia multipla.

Il presente lavoro descrive un possibile approccio, basato sullo sviluppo di applicazioni client-server, per la remotizzazione del laboratorio didattico di Misure Elettroniche. Per essere più precisi quello che sarà reso disponibile agli utenti, che in questo caso potrebbero essere gli studenti del corso di misure, è la possibilità di effettuare da remoto delle misurazioni su multimetro digitale.

Il laboratorio è realizzato in un'aula didattica appositamente attrezzata di multimetro digitale (Hewlett Packard 34401a) e basetta elettronica per le misurazioni. A questi si aggiunge un PC server al quale gli strumenti sono fisicamente collegati utilizzando un'interfaccia standard GPIB (General Purpose Interface Bus). Il controllo remoto degli strumenti può essere effettuato direttamente dal server, oppure attraverso il protocollo TCP/IP da parte di altri utenti collegati in rete. Tali utenti possono essere studenti che si trovano al di fuori del laboratorio (ad esempio nelle loro abitazioni) e che sono collegati al PC server via web.

A livello software, lato server implementeremo un Web service sfruttando l'ambiente di sviluppo software LabVIEW di National Instrument; lato client utilizzeremo la piattaforma javaFX per realizzare un'interfaccia grafica del nostro applicativo.

Le possibilità offerte da LabVIEW di realizzare web service ci consentiranno di comunicare da remoto usando un client personalizzato (costum client), e di sfruttare i vantaggi del protocollo HTTP.

Concludendo, possiamo suddividere il progetto in tre blocchi come segue:

- **Server**: svilupperemo un Web service per misure con multimetro utilizzando il software LabVIEW.
- **Client**: realizzeremo un'applicazione web che consenta di interagire da remoto con il server LabVIEW. Per lo sviluppo utilizzeremo la piattaforma javaFX.
- **Comunicazione client/server**: Comunicheremo tra dispositivi client e server con il protocollo HTTP.

Nei prossimi capitoli presenteremo il lavoro suddividendolo come segue:

- Breve panoramica sui sistemi client-server.
- Introduzione ai Web service ed in particolare allo sviluppo di Web service in LabVIEW.
  - Web service SOMMA in LabVIEW
  - Web service misura con multimetro in LabVIEW
  - Web service misure ripetute con multimetro in LabVIEW
- Creazione di client per la somma e la misura con multimetro sfruttando la tecnica AJAX.
- La piattaforma JavaFX: introduzione al linguaggio javaFX script e sviluppo dei seguenti client:

  - Client per la SOMMA in JavaFX
  - Client misura con multimetro in JavaFX
  - Client misure ripetute con multimetro in JavaFX

- Gestione di piú client: vedremo come tenere traccia degli utenti che accedono al servizio utilizzando le sessioni e i cookie.

CAPITOLO

2

CLIENT-SERVER

La maggior parte dei servizi telematici offerti da internet si basano su una particolare modalità di interazione denominata tecnicamente architettura client-server. Questo nome indica un'architettura software che è costituita da due moduli integrati ma distinti, residenti generalmente su calcolatori diversi. Chi inizia la comunicazione, per la richiesta di uno specifico servizio, viene denominato *client* (cliente), mentre chi elabora le richieste in arrivo e restituisce una risposta, è il *server* (serviente).

Più semplicemente, i sistemi client/server sono un'evoluzione dei sistemi basati sulla condivisione semplice delle risorse. La presenza di un server permette ad un certo numero di client di condividerne le risorse, lasciando che sia il server a gestire gli accessi alle risorse per evitare i conflitti tipici dei primi sistemi informatici.

Sono sempre di più i software, come il web, l'e-mail, i database, che sono divisi in una parte client (residente ed in esecuzione sul pc client) ed una parte server (residente ed in esecuzione sul server).

Il software server, oltre alla gestione logica del sistema, deve implementare tutte le tecniche di gestione degli accessi, allocazione e rilascio delle risorse, condivisione e sicurezza dei dati. Per fare un esempio, un server di posta elettronica è paragonabile ad un qualunque ufficio postale. Gli utilizzatori, per accedere via client alla loro cassetta di posta elettronica, devono essere

stati autorizzati. In modo analogo, un utente deve possedere la chiave della cassetta di un ufficio postale dalla quale vuole prelevare la corrispondenza.

In un ambiente client-server, sul computer client, è in esecuzione un software applicativo, il quale:

- Abilita l'utente a spedire una richiesta ad un server web.
- Formatta la richiesta in modo che il server possa capirla.
- Formatta la risposta del server in modo che l'utente possa leggerla.

Lato server, è in esecuzione un software il quale:

- Riceve la richiesta da un client e la processa
- Risponde spedendo l'informazione richiesta al client.

Affinchè l'interazione tra client e server possa essere effettuata, è necessario che entrambi utilizzino un linguaggio comune, ovvero un protocollo di comunicazione. Con questo termine si intende l'insieme di regole, formati e procedure che le due entità devono rispettare allo scopo di scambiare dati correttamente. Su internet vengono utilizzati numerosi protocolli, specifici per ogni servizio, per essere piú chiari facciamo qualche semplice esempio:

- Un utente collegato in rete utilizzando un comune browser (Internet Explorer, Mozilla Firefox, Safari) comunica con un server web grazie al protocollo HTTP.
- Un utente che si collega ad un server di posta utilizzando un comune mail user agent (Outlook, Mozilla Tunderbird), comunica utilizzando i protocolli SMTP e POP.
- Un utente che utilizza un software client come (FileZilla, JDownloader) può trasferire file ad un server FTP attraverso il protocollo FTP.

Per i nostri scopi focalizzeremo l'attenzione sul protocollo HTTP, uno tra i piú utilizzati per trasferire informazioni sul Web. HTTP si basa su un protocollo di tipo client-server che consente a due processi di comunicare usando una connessione TCP/IP.

La Internet protocol suite, più nota come TCP/IP, è un'architettura di protocolli stratificata strutturata nei seguenti tre livelli:

- Application (strato applicazione).
- Transport (strato di trasporto).

- Internet (strato di rete).

Al livello più alto della gerarchia, strato applicazione, appartengono i protocolli che forniscono direttamente dei servizi agli utenti e tra questi ritroviamo HTTP (Hyper Text Transfer Protocol), per il trasferimento di pagine web.

HTTP funziona su un meccanismo richiesta/risposta; il client esegue una richiesta ed il server restituisce la risposta. Differisce da altri protocolli di livello applicazione (FTP, SMTP), per il fatto che le connessioni vengono generalmente chiuse una volta che una particolare richiesta (o una serie di richieste correlate) è stata soddisfatta. Questa caratteristica lo rende ideale per il web, dove molto spesso, le pagine contengono collegamenti (link) a pagine ospitate da altri server.

Per la comunicazione sono definiti 8 metodi mediante i quali il client può specificare l'azione che desidera eseguire sulla risorsa web. I metodi sono: GET, POST, HEAD, PUT, DELETE, TRACE e OPTIONS. I più usati tuttavia sono solamente due, GET e POST.

- **GET**: permette di recuperare informazioni dal server. Il metodo GET consiste nell'accodare i dati all'indirizzo della pagina richiesta; facendo seguire il nome della pagina da un punto interrogativo e dalle coppie nome/valore dei dati che ci interessano. Nome e valore sono separati da un segno di uguale mentre le diverse coppie nome/valore sono divise dal simbolo &. La stringa che si trova dopo il punto interrogativo, contenente nomi e valori dei parametri, viene detta query string.

- **POST**: viene usato per richiedere al server di accettare l'entità inclusa nella richiesta come nuova condizione per la risorsa individuata dall'URL. Tipicamente POST viene usato per invio di dati complessi (esempio la sottoscrizione di un servizio).

Per completezza di analisi, riportiamo di seguito la sintassi usata da HTTP per identificare una risorsa web (ovvero la sintassi URL). Essa è specificata dalla raccomandazione RFC2616.

http://host:port/path?parameters

- **host**: identifica il dispositivo su cui è in esecuzione il server (es: indirizzo IP del computer).

- **port**: è il numero della porta associata a tale servizio (per default il numero è 80 ma può essere cambiato se tale valore è già assegnato ad un'altra risorsa).

- **path**: è il percorso che identifica il server all'interno dell'host.
- **parameters**: è la lista dei parametri associati alla risorsa. Per ogni parametro deve essere specificato l'identificativo o chiave (key) ed il valore, separati dal carattere "=". Per separare i vari parametri viene usato il simbolo "&". Esempio: "a=3,12&b=6,43".

Concludendo, nell'ambito delle applicazioni client-server molte sono le possibilità per l'implementazione software sia lato server che lato client.

Per i nostri scopi, lato client svilupperemo l'interfaccia grafica del nostro applicativo utilizzando il linguaggio javaFX, mentre lato server, il controllo della strumentazione e l'elaborazione dei dati acquisiti, sarà ottenuto utilizzando l'ambiente di sviluppo software LabVIEW.

Le possibilità offerte da LabVIEW di realizzare Web Services ci consentiranno anzitutto di gestire il nostro applicativo usando un client personalizzato javaFX, ed inoltre di comunicare sfruttando i vantaggi offerti dal protocollo HTTP.

CAPITOLO

3

WEB SERVICE

3.1 Cosa sono i Web service

Un Web service è un componente applicativo. Possiamo definirlo come un sistema software in grado di mettersi al servizio di un applicazione comunicando su di una medesima rete tramite il protocollo HTTP. Un Web service consente quindi alle applicazioni che vi si collegano di usufruire delle funzioni che mette a disposizione.

Per fare un esempio potremmo ipotizzare un Web service che chiameremo cambiavalute. Il nostro Web service fornisce le seguenti operazioni: cambio euro/dollaro e viceversa. Questo Web service potrebbe essere offerto da un istituto bancario ed una nostra applicazione potrebbe utilizzarlo per effettuare le operazioni di cambio senza doversi preoccupare dei tassi in vigore al momento dell'operazione.

Già dopo questo primo esempio si può notare come le operazioni svolte da un Web service non sono nulla di eclatante, qualsiasi comune applicazione potrebbe infatti effettuare l'operazione di cambio. Ciò che una comune applicazione però non può fare è mettersi in comunicazione con un altro software come ha fatto cambiavalute nel nostro esempio.

Un Web service è in grado di offrire un'interfaccia software assieme alla descrizione delle sue caratteristiche, cioè è in grado di farci sapere che

funzioni mette a disposizione (senza bisogno di conoscerle a priori) e ci permette inoltre di capire come vanno utilizzate. Ciò significa che (sempre per rimanere al nostro esempio) con una semplice connessione a cambiavalute, anche senza conoscerlo, possiamo stabilire le operazioni che fornisce e possiamo subito iniziare ad usarle perchè ogni operazione ha una sua descrizione comprendente i parametri che si aspetta di ricevere, quelli che restituirà ed il tipo di entrambi.

3.2 Perchè utilizzare i Web service

La ragione principale per la creazione e l'utilizzo di Web Service è il disaccoppiamento che l'interfaccia standard esposta dal Web Service rende possibile fra il sistema utente ed il Web Service stesso. Modifiche ad una o all'altra delle applicazioni possono essere attuate in maniera trasparente all'interfaccia tra i due sistemi; tale flessibilità consente la creazione di sistemi software complessi, costituiti da componenti svincolati l'uno dall'altro e consente una forte riusabilità di codice ed applicazioni già sviluppate.

Ecco riassunti i principali motivi per utilizzare o sviluppare un Web service.

- I Web service permettono l'interoperabilità tra diverse applicazioni software e su diverse piattaforme hardware/software.
- Utilizzano un formato dei dati di tipo testuale, quindi più comprensibile e più facile da utilizzare per gli sviluppatori.
- Normalmente, essendo basati sul protocollo HTTP, non richiedono modifiche alle regole di sicurezza utilizzate come filtro dai firewall.
- Permettono di riutilizzare applicazioni già sviluppate.
- Fintanto che l'interfaccia rimane costante, le modifiche effettuate ai servizi rimangono trasparenti.
- I servizi web sono in grado di pubblicare le loro funzioni e di scambiare dati con il resto del mondo.
- Tutte le informazioni vengono scambiate attraverso protocolli aperti.

3.3 Web service RESTful

REST, Representational State Transfer, è uno stile architetturale per sistemi software distribuiti. Il termine è stato introdotto e definito nel 2000 da **Roy Fielding**, uno dei principali autori delle specifhe del protocollo HTTP. Indica una serie di principi architetturali per la progettazione di Web Service.

Concetto centrale per un sistema RESTful è quello di risorsa. Una risorsa è qualunque entità che possa essere indirizzabile tramite Web, cioè accessibile e trasferibile tra client e server.

- Un articolo di un sito di notizie.
- Un'immagine di una web gallery.
- Uno studente di una qualche universita.

Inizialmente REST venne descritto da Fielding nel contesto del protocollo HTTP, il protocollo a livello Applicazione maggiormente utilizzato; ma un sistema RESTful si può appoggiare ad un qualunque altro protocollo che fornisca un vocabolario altrettando ricco.

Perchè un Web Service sia conforme alle specifche REST deve avere alcune specifche caratteristiche:

- Architettura basata su client e server.
- **stateless**. Ogni ciclo di request/response deve rappresentare un'interazione completa del client con il server. In questo modo non è necessario mantenere informazioni sulla sessione utente, minimizzando l'uso di memoria del server e la sua complessità.
- **uniformemente accessibile**. Ogni risorsa deve avere un'indirizzo univoco e ogni risorsa di ogni sistema presenta la stessa interfaccia, precisamente quella individuata dal protocollo HTTP.

3.4 Web Service in LabVIEW

LabVIEW (abbreviazione di Laboratory Virtual Instrumentation Engineering Workbench) è un ambiente di sviluppo integrato per il linguaggio di programmazione visuale di National Instrument.

Dalla versione LabVIEW 5.1 è possibile implementare un Web-server che consente di operare da remoto, sul pannello frontale di una macchina, usando un comune browser. Il pannello frontale del VI (Virtual Instrument)

viene caricato come una pagina html dinamica della quale deve essere fornito l'URL (Uniform Resource Locator). Per accedere al pannello remoto è quindi possibile utilizzare un comune web browser, oppure un client in LabVIEW, facendo attenzione che nel caso di un comune web browser, è necessario installare LabVIEW Run-Time Engine (fornito gratuitamente dalla National Instrument).

Con **LabVIEW 8.6** è resa disponibile una diversa modalità per la gestione remota del VI. Questa modalità prende il nome di **Web Service** e si basa su concetti completamente differenti da quelli utilizzati per la creazione di pannelli remoti.

I vantaggi offerti da questa nuova soluzione sono molteplici, riassumiamo di seguito i più importanti:

- E possibile comunicare con applicazioni Labview da qualsiasi dispositivo in grado di connettersi al Web (anche un telefono cellulare o un microcontrollore dotato di interfaccia di rete Ethernet, ZigBee, BlueTooth, WiFi, ecc).

- La comunicazione avviene usando il protocollo HTTP e non un protocollo proprietario (non è più necessario installare LabVIEW Run-Time Engine).

- Applicazioni LabVIEW possono essere controllate da remoto usando client personalizzati (custom client).

- E possibile gestire qualsiasi tipo di dato MIME (Multipurpose Internet Mail Extensions), così come testi, immagini e video.

3.5 Come creare Web service in LabVIEW

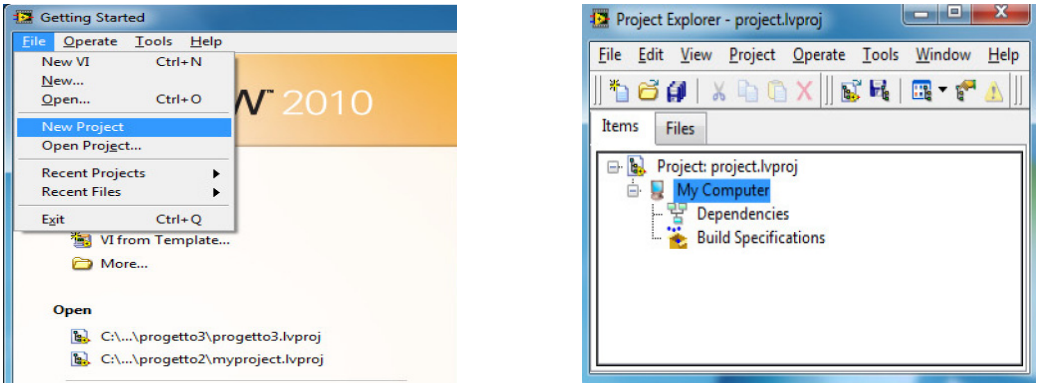
Presenteremo in questo paragrafo come creare un Web service utilizzando LabVIEW. Lo faremo realizzando due semplici servizi web di esempio, il primo per la somma di due numeri impostati dall'utente, il secondo per una misura di resistenza con multimetro, descrivendo passo passo tutte le operazioni necessarie.

3.5.1 Web service SOMMA in LabVIEW

Per configurare un VI in modo tale da poter essere gestito da remoto, è necessario come prima cosa inserire il VI in un progetto:

- Dalla pagina iniziale di LabVIEW (**Getting started**) creare un progetto vuoto e salvarlo con il nome desiderato (esempio: project.lvproj).

- Il progetto viene gestito attraverso i meccanismi disponibili nella sezione **Project Explorer** (che viene aperta automaticamente alla creazione di un nuovo progetto).



(a) New project

(b) Sezione project explorer

Figura 3.1: Creazione di un nuovo progetto LabVIEW

All'interno del nostro nuovo progetto, bisogna creare un VI dove poter realizzare il programma LabVIEW. Cliccando con il tasto destro del mouse su **MyComputer** si apre un menú a tendina, scegliere **New→VI** per aggiungere un nuovo VI al progetto.

Apriamo il VI creato e lavoriamoci come segue:

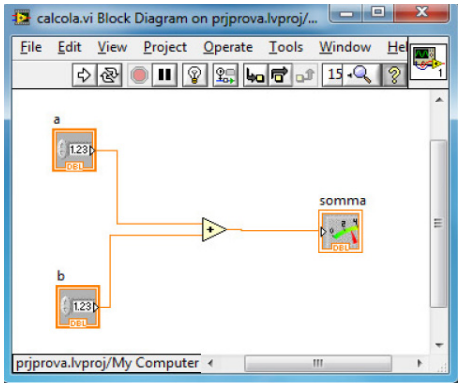
- Nel **block diagram** inserire il codice necessario per eseguire la somma tra due numeri **a** e **b** di tipo double, ed inviare il risultato all'indicatore **somma**.

- Creare il **pannello frontale** con i due controlli a e b e l'indicatore somma. Si veda come esempio la figura 3.2.

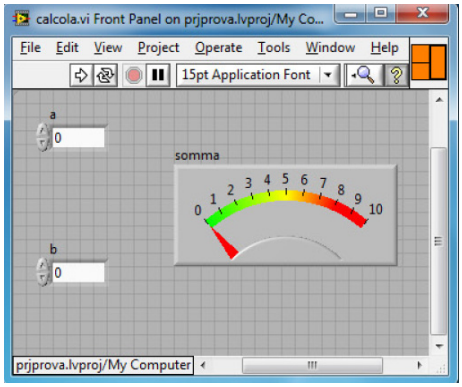
- Cliccando con il tasto destro del mouse sull'icona connettore, si apre un menú a tendina, selezionare il **patterns** con tre pin.

- Collegare i pin del connettore rispettivamente ai due controlli e all'indicatore.

- Salvare il VI (esempio: calcola.vi)



(a) Diagramma a blocchi



(b) Pannello frontale

Figura 3.2: Creazione del VI SommaAB

Arrivati a questo punto abbiamo tutto il necessario per configurare e distribuire il nostro primo servizio web.

Sulla finestra di Project Explorer, cliccare con il tasto destro del mouse su **Build Specification** e scegliere **New→Web Service RESTful**. Con la creazione del Web Service viene aperta la finestra **My Web Service Properties** mediante la quale configurare il servizio.

Come mostrato in figura 3.3(a), le proprietà da impostare si suddividono in otto categorie, per i nostri scopi sarà sufficiente soffermarci sulle prime tre della lista: Information, Source Files e URL Mappings.

1. Information

   - La prima informazione da specificare nel menú Information è il **Build Specification Name**. Questo associa un nome al Web service che si sta creando(il nome viene usato per creare la directory di destinazione).
   - La seconda informazione è il **Service Name** che viene usato per costruire l'URL mediante il quale identificare la risorsa in internet.

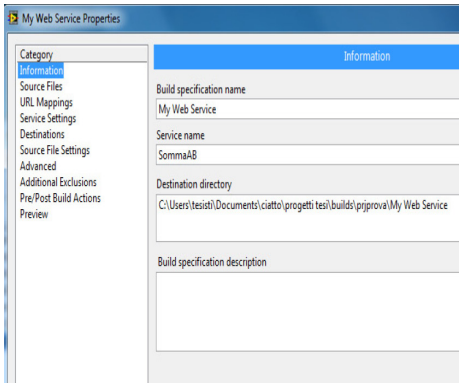
2. Source Files

   - Dal menú Source Files si possono selezionare i VI da associare al Web Services.
   - Per ogni VI selezionato è necessario configurare il tipo di uscita prodotta, abbiamo due scelte:

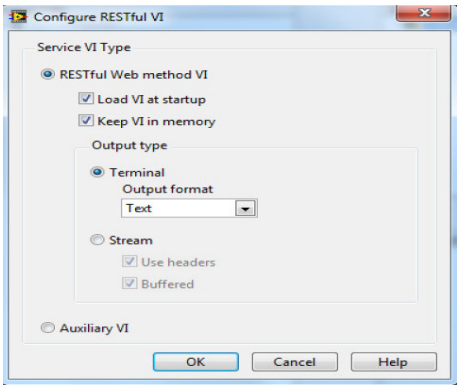
     (a) **da terminale**: in questo caso scegliere anche il formato dell'uscita tra i seguenti (text, html, xml)

(b) **stream**

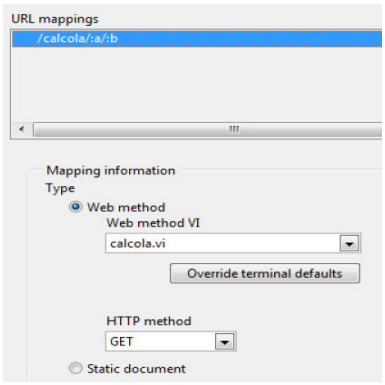
- Per il nostro esempio, impostare l'uscita da terminale con formato di tipo **text**.



(a) Information



(b) Service name



(c) URL Mappings

Figura 3.3: My web service properties

3. URL Mappings

- Dal menú URL Mappings, associare ad ogni VI eseguito come Web Service, il corrispondente URL.
- Per collegare gli ingressi dell'URL ai terminali di ingresso del VI è necessario inserire la stringa ":nome_terminale" alla fine dell'URL. Ogni terminale deve essere separato con una barra "/".
- Dalla figura 3.3(c) si vede che **calcola** è il nome scelto per il servizio, mentre **a** e **b**, essendo i nomi associati ai terminali di ingresso del VI, sono stati riportati come descritto al punto precedente.

- Per ultimo bisogna specificare il modo in cui i dati arriveranno al VI. I metodi piú utilizzati sono GET e POST, per il nostro esempio usiamo il GET.
- Chiudere la finestra cliccando su OK. A questo punto manca solo da compilare e distribuire il servizio.

Dalla finestra di **Project Explorer**, cliccare con il tasto destro del mouse sul Web service creato (esempio: Somma A+B Web Service) e compilare mediante l'opzione **build**. Se la compilazione viene eseguita con successo, ripetere l'operazione cliccando su **deploy**.

Se non sono comparsi messaggi di errore, siamo pronti per testare il lavoro. Avviare un browser e digitare il comando:

http://127.0.0.1:8080/SommaAB/calcola/10/8

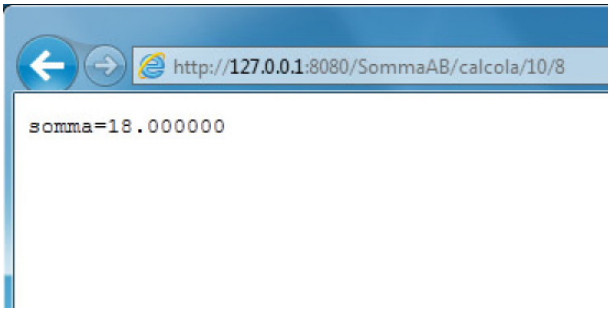


Figura 3.4: esempio di somma usando un brawser web

Analizzando la stringa HTTP digitata sul nostro web browser, si può vedere come le operazioni di configurazione fatte poco fa sono state utilizzate nella chiamata al servizio. La somma richiesta viene visualizzata su pagina web come riportato in figura 3.4.

Web service SOMMA sfruttando HTTP

Le funzionalità appena presentate sono per molti aspetti limitate e un migliore e più efficiente utilizzo delle potenzialità offerte da LabVIEW Web Services si può raggiungere. Ciò è reso possibile da HTTP (Hypertext Transfer Protocol), uno tra i protocolli piú utilizzati per trasferire informazioni sul web.

A questo scopo si vuole realizzare un Web service con le stesse funzionalità del precedente, con la differenza che la riga di comando usata per interagire con il VI è del tipo:

http://host:port/path?a=valore&b=valore

Per creare un Web service capace di interpretare comandi HTTP in questa forma, il VI deve essere realizzato come in figura 3.5.

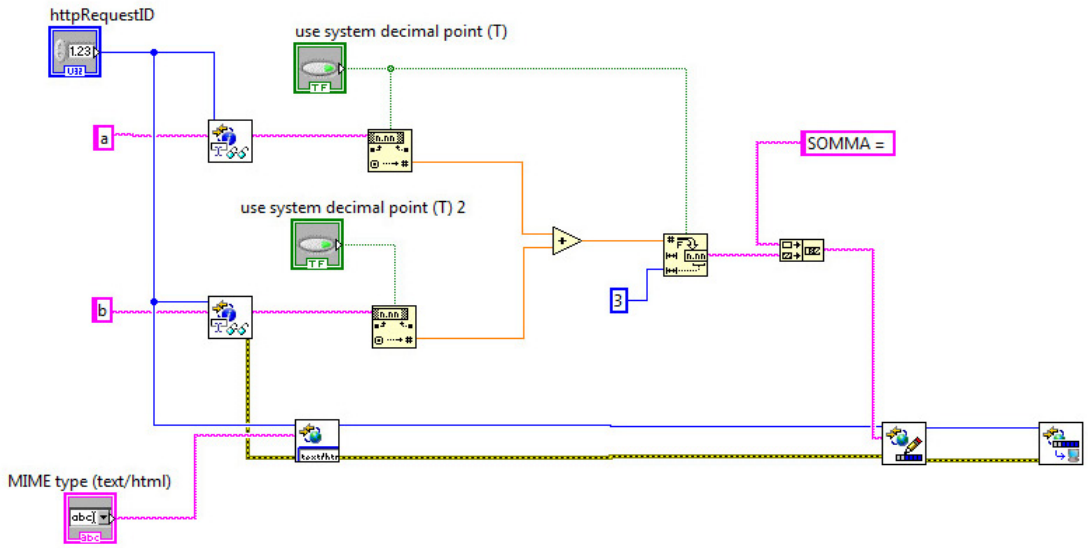


Figura 3.5: VI per la somma di due numeri

Analizziamo il programma piú in dettaglio:

- Il VI ha un solo terminale associato all'unico controllo **httpRequestID**. Questo controllo deve essere connesso ad uno dei pin del connettore.
- Cliccare con il tasto destro del mouse sull'icona connettore e selezionare il **patterns** da uno, collegarlo quindi al controllo httpRequestID.
- I parametri a e b specificati nella stringa HTTP, vengono letti per mezzo del VI **Read Form Data**. Questo restituisce una stringa per ciascun valore.
- I valori associati ai parametri a e b vengono convertiti in numeri, sommati, riconvertiti in stringa ed inviati al buffer in uscita gestito dal VI **Write Response**.
- Il formato della risposta inviata al client viene impostata mediante il VI **Set HTTP Response MIME Type**. Nel nostro esempio viene scelto il formato html.

- Tramite il VI **Flush Output** viene inviata la risposta al client.

Per quanto riguarda la configurazione del servizio resta tutto come prima, eccetto qualche passaggio che riportiamo:

- L'uscita del VI deve essere di tipo **stream**.
- Non essendoci terminali di ingresso in grado di ricevere valori da remoto, l'URL è composto dal solo nome della risorsa. Per questo secondo esempio abbiamo scelto **somma**.

Avviare un browser e provare il nuovo Web service creato con il comando che segue, il risultato è riportato in figura 3.6.

http://127.0.0.1:8080/SommaAB/somma?a=5&b=3

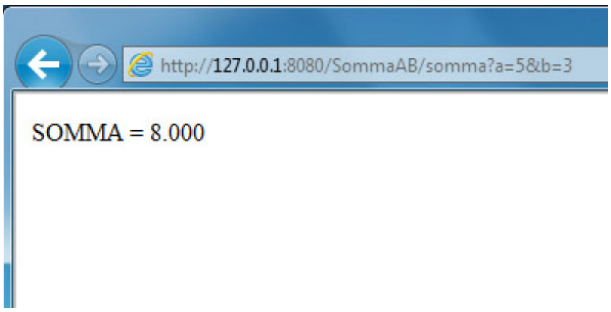


Figura 3.6: esempio di somma usando un brawser web

Questo modo di procedere lo utilizzeremo in tutti i nostri progetti, comunicheremo tra i nostri dispositivi client e server sfruttando i vantaggi offerti dal protocollo HTTP.

3.5.2 Web service misura con multimetro in LabVIEW

Questo paragrafo ha lo scopo di spiegare come realizzare un Web service che offra la possibilità ad un utente di fare delle misurazioni da remoto su multimetro digitale (Hewlett Packard 34401a).

Quello che sarà chiesto al nostro programma LabVIEW è comunicare con il multimetro digitale, eseguire la misura scelta e inviare il risultato al cliente in attesa. Il cliente sceglierà il tipo di misura con il seguente criterio:

- **0** resistenza

- **1** tensioneDC
- **2** tensioneAC
- **3** correnteDC
- **4** correnteAC

Il nostro Web service effettuerà la misura in base al valore impostato dall'utente per il parametro di ingresso. Per fare un esempio, la seguente riga di comando sarà interpretata come una richiesta di misura su resistenza:

http://127.0.0.1:8080/project2/multimetro?a=0

Analizziamo il VI per la misura con multimetro, figura 3.7.

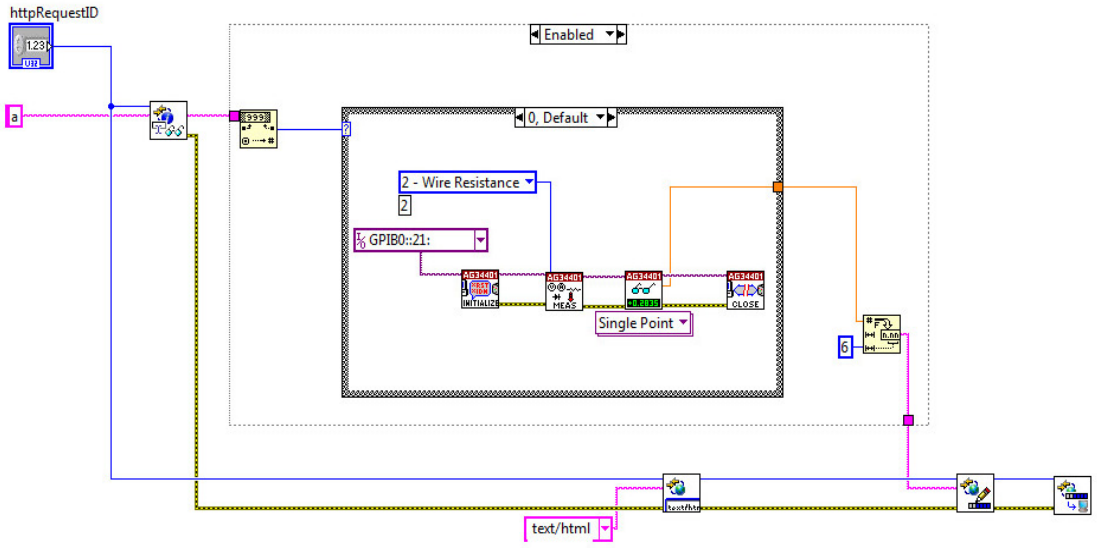


Figura 3.7: VI per la misura con multimetro

Anche in questo caso abbiamo un solo terminale associato all'unico controllo **httpRequestID**. Questo controllo viene connesso ad uno dei pin del connettore.

Il parametro **a**, indicativo della misura scelta dall'utente, viene letto per mezzo del VI **Read From Data**. Il valore associato ad **a**, viene convertito in numero ed inviato ad una struttura di tipo **case**. Qui è presente il codice necessario per la misura con multimetro digitale.

Al fine di poter comunicare correttamente, abbiamo utilizzato gli **instrumet driver** propri del multimetro digitale (Hewlett Packard 34401a). Questi sono messi a disposizione degli utenti, e scaricabili gratuitamente attravero la Instrument Driver Network di National Instrument. Presentiamo di seguito i blocchi utilizzati:

- **Initialize** è necessario per stabilire una comunicazione con lo strumento. Come parametro in ingresso riceve il VISA resurce name. Per funzionare, lo strumento dev'essere già stato collegato e correttamente rilevato dal computer.

- **Configure Measurement** serve per impostare il tipo di misura scelta (resistenza, tensioneDC, tensioneAC, ecc). Questa impostazione cambia in funzione del valore associato al parametro **a**.

- **Read(Single Point)** legge e ritorna una singola misura dallo strumento. Il formato di ritorno è di tipo double.

- **Close** termina la connessione con lo strumento. Questo blocco deve essere sempre presente al fine di chiudere la comunicazione correttamente.

Il valore misurato da multimetro viene convertito da numero a stringa ed inviato al buffer in uscita gestito da VI **Write Response**. Il formato della risposta inviata viene impostata mediante il blocco **Set HTTP Response MIME Type**.

Per testare il nostro nuovo Web Service operiamo come prima. Tramite browser web digitiamo il comando per una misura su resistenza, il risultato è riportato in figura 3.8.

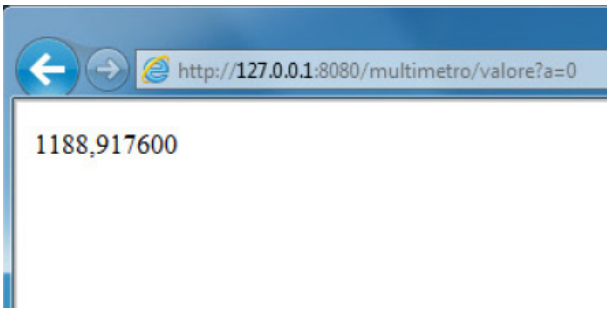


Figura 3.8: esempio di misura su resistenza con web brawser

3.5.3 Web service misure ripetute con multimetro in LabVIEW

Nel paragrafo precedente abbiamo visto come realizzare un Web service per la misura con multimetro. In questa sezione amplieremo il lavoro in modo che l'utente possa effettuare un numero n di misure ripetute.

Il nostro programma LabVIEW sarà strutturato come segue:

- Il valore associato al parametro di ingresso **a**, selezionerà il tipo di misura. In modo analogo, il parametro **b** imposterà il numero **n** di misure.

- Il vettore con le **n** misure effettuate viene spedito in ingresso ad un ciclo for. Il ciclo va da 0 a n (numero di misure o dimensione del vettore di misure).

- Il ciclo for ha lo scopo di accodare le misure una all'altra a formare una stringa (query string).

- La stringa contenente tutti i dati viene inviata all'utente.

- Per fare in modo che il client recuperi ogni singola misura in modo corretto, viene interposto uno spazio vuoto tra la fine di una misura e l'inizio della successiva.

Per quanto riguarda il codice necessario alla comunicazione con il multimetro digitale, rimandiamo il lettore al paragrafo precedente, tutto è stato spiegato nel dettaglio.

Quello che abbiamo fatto di nuovo, è ideare un ciclo for per accodare una all'altra le **n** misure a formare una stringa alfanumerica. Questo consentirà di inviare tutti i dati al client.

Vediamo in figura 3.9 il VI per la creazione del Web service misure ripetute con multimetro, analizziamone le parti principali:

- **Read(Multiple Points)** legge e ritorna un vettore di n misure dallo strumento. Il numero di misurazioni viene impostato inviando il valore del parametro di ingresso **b** al terminale **sample count** del VI in questione.

- L'array di misure viene inviato in ingresso al ciclo for. Questo forza il numero di iterazioni alla dimensione dell'array (tunnel con indicizzazione abilitata).

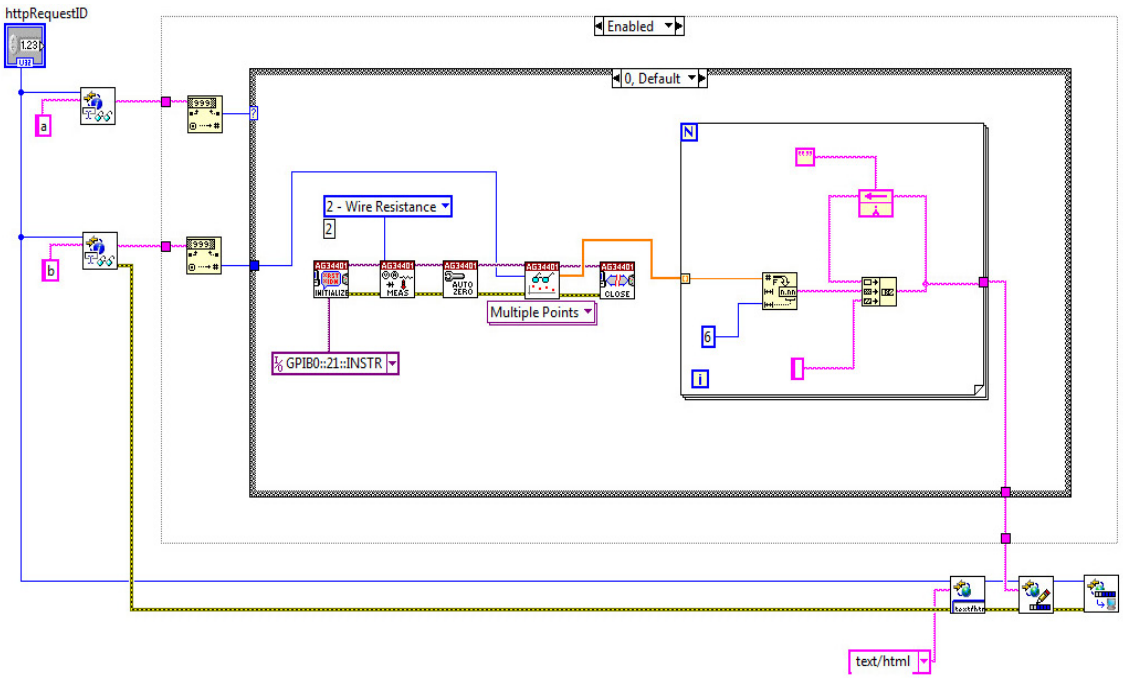


Figura 3.9: VI per misure ripetute con multimetro

- Le n misure vengono accodate una all'altra passando i valori da un'iterazione del ciclo alla successiva. Per fare questo si utilizza una **feedback note** che viene creata automaticamente quando si collega l'uscita di un blocco con l'ingresso dello stesso.

- La stringa completa con le n misure, viene fornita in ucita al ciclo for e spedita in ingresso al VI **Write Response**. Per fare in modo che in uscita venga fornito solo l'ultimo valore calcolato (concatenazione delle n misure), l'indicizzazione viene disabilitata.

Per utilizzare da remoto il web service, sarà necessario sviluppare un software client in grado di comunicare correttamente con il nostro programma LabVIEW. A tal fine sfrutteremo JavaFX, un software sviluppato da Sun per la creazione di rich internet application.

CAPITOLO

4

CREAZIONE DI CLIENT IN AJAX

Introduzione alla tecnica AJAX

L'acronimo AJAX, che significa esattamente Asynchronous JavaScript And XML (JavaScript asincrono ed XML), è stato enunciato per la prima volta da Jesse Garrett, nel 18 Febbraio 2005, come titolo di un post all'interno del suo blog.

Il concetto è in parte espresso nell'acronimo scelto, un utilizzo asincrono di Javascript che attraverso l'interfacciamento con XML, può permettere ad un client di richiamare informazioni lato server in modo veloce e trasparente, allargando gli orizzonti delle rich internet applications. Queste applicazioni fino a poco tempo fa erano legate principalmente alle tecnologie Adobe-Macromedia Flash o Java (con le applet). Entrambe purtroppo non sempre interpretabili dai client degli utenti e troppo spesso usate a sproposito con il solo scopo di stupire.

Se parliamo di AJAX, oggi, parliamo di un oggetto specifico: XMLHttpRequest. A seconda del browser usato prende nomi differenti o viene richiamato in maniera differente. Questo oggetto permette di effettuare la richiesta di una risorsa (con HTTP) ad un server web in modo indipendente dal browser. La richiesta è asincrona, il che significa che non bisogna necessariamente

attendere che sia stata ultimata per effettuare altre operazioni, stravolgendo sotto diversi punti di vista il flusso dati tipico di una pagina web.

4.1 Client per la somma di due numeri

Nei paragrafi precedenti abbiamo visto come realizzare un Web service per la somma di due numeri digitati dall'utente. Per questo scopo il metodo piú semplice per l'interrogazione del web server da remoto è utilizzare direttamente la stringa di comando (URL) cambiando i valori associati ai parametri **a** e **b**. Il valore ricevuto viene visualizzato su pagina web.

Gestire un applicativo da remoto direttamente mediante comandi HTTP è tuttavia un metodo assolutamente no-user friendly. Un migliore approccio consiste nel realizzare un client adeguato, che permetta all'utente di effettuare le operazioni in modo semplice e visualizzare i risultati in modo chiaro.

Come esempio per iniziare, realizzeremo una pagina HTML dinamica. I valori dei parametri **a** e **b** vengono inseriti mediante un form HTML e letti da una funzione scritta usando il linguaggio JavaScript.

Quando l'utente preme il pulsante "calcola" viene generato un evento in seguito al quale viene inviato un comando HTTP di tipo GET. La risposta, gestita mediante la tecnica AJAX, sarà visualizzata nell'area di testo etichettata Somma.

Il primo passo nella realizzazione del client, sarà creare un semplice form HTML. Il form, chiamato **myForm**, è composto da 4 oggetti: tre oggetti di tipo "input" di testo, per l'inserimento dei parametri **a** e **b** e la visualizzazione del risultato, e un oggetto di tipo "button" per l'invio del comando.

Si è scelto di utilizzare un oggetto di tipo button, per lasciare la possibilità all'utente di controllare i parametri impostati prima di inviarli al server. Quando l'utente preme il bottone, viene invocata la funzione JavaScript: **laodXMLDoc()**.

- Questa funzione viene inserita nella pagina HTML in un'area identificata dai due tag: <script></script>.
- Il linguaggio degli script viene specificato mediante il parametro type che in questo caso viene impostato come text/javascript.
- Il codice della funzione viene racchiuso tra due parentesi graffe .

Al fine di dialogare con il server, è necessario creare un oggetto di tipo **XMLHttpRequest**. Per mantenere la compatibilità tra i diversi browser

Listing 4.1: creazione dell'oggetto XMLHttpRequest

```
if(window.XMLHttpRequest){
 //codice per IE7, Firefox, Chrome, Opera, Safari
 xmlhttp=new XMLHttpRequest();
 }
  else{ //codice per IE7, IE5
 xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
 }
```

bisogna prevedere due diversi meccanismi come illustrato nel frammento di codice riportato.

Definito l'oggetto, è indispensabile conoscerne metodi e parametri per sfruttarlo al meglio a seconda delle necessità. Riportiamo una lista, in ordine alfabetico, dei metodi supportati da tutti i browsers AJAX compatibili:

- abort
- getAllResponseHeaders
- getResponseHeader
- open
- send
- setRequestHeader

Questi metodi permettono di realizzare le varie operazioni asincrone; vediamo quelli di interesse per il nostro progetto.

Open viene invocato per impostare un comando HTTP da inviare al web server. Il metodo è definito per accettare 5 parametri, noi ne useremo tre.

Listing 4.2: sintassi del metodo open

```
xmlhttp.open("method","http://127.0.0.1:8080/path?
              a="value"&b="value,true);
```

Il primo parametro è una stringa che indica il metodo di invio dati. Possiamo valorizzarlo come get o post. La differenza principale tra questi due metodi è la stessa di un normale form HTML: scegliendo GET le variabili verranno appese alla pagina selezionata (per esempio pagina.html?variabile=valore),

mentre utilizzando POST queste verranno inviate in modo invisibile all'utente e senza i limiti tipici del GET.

Il secondo parametro è il nome della pagina da leggere, e sarà rappresentato da una comune stringa HTTP.

Per ultimo, il terzo parametro, specifica un valore booleano che deve essere impostato come **true**, vero, per indicare al metodo open che la richiesta da effettuare è di tipo asincrona. Come già spiegato, asincrona significa che il client non rimane in attesa di una risposta dal server, ma torna a svolgere altre funzioni e processerà la risposta del server quando questa arriva.

Per inviare il comando HTTP impostato con open() al web server, si utilizza il metodo **send()**.

Ci resta da analizzare in che modo il nostro software client gestisce i dati in arrivo dal server. Per questo scopo è necessario definire una funzione che verrà eseguita quando il client riceve una risposta dal server (evento **onreadystatechange**).

Listing 4.3: fuzione per l'evento onreadystatechange

```
1 xmlhttp.onreadystatechange=function(){
2  if(xmlhttp.readyState==4 & xmlhttp.status==200){
3    document.myForm.result.value=xmlhttp.responseText;
4    }
```

Quello che fa la funzione, è controllare lo stato della richiesta verificando il valore di alcuni **parametri**, e successivamente, nel caso in cui l'azione è stata eseguita con successo, copiare i risultati ricevuti nel form HTML.

La lista dei parametri comunemente supportati dai vari browsers è la seguente, in ordine alfabetico:

- onreadystatechange
- readyState
- responseText
- responseXML
- statusText

Il primo e fondamentale parametro da considerare durante uno scambio dati in AJAX è **readyState**. È una varibile di tipo intero, con valori che vanno da 0 a 4. Gli stati che la richiesta può assumere sono :

- **0 Uninitialized** l'oggetto XMLHttpRequest esiste, ma non è stato richiamato alcun metodo per inizializzare una comunicazione.

- **1 Open** è stato richiamato il metodo open() ed il metodo send() non ha ancora effettuato l'invio dati.

- **2 Sent** il metodo send() è stato eseguito ed ha effettuato la richiesta.

- **3 Receiving** i dati in risposta cominciano ad essere letti.

- **4 Loaded** l'operazione è stata completata.

Il secondo parametro di interesse è **status**. Status contiene un codice di tre cifre che descrive lo stato della richiesta inviata al server. **200 OK** è la risposta standard per le richieste HTTP andate a buon fine.

- **1xx Informational**: richiesta ricevuta, continua l'elaborazione.

- **2xx Success**: l'azione è stata ricevuta con successo, compresa ed accettata.

- **3xx Redirezione**: il client deve eseguire ulteriori azioni per soddisfare la richiesta.

- **4xx Client error**: la richiesta è sintatticamente scorretta o non può essere soddisfatta.

- **5xx Server Error**: il server ha fallito nel soddisfare una richiesta apparentemente valida

Per ultimo vediamo **responseText**. Questo parametro conterrà i dati restituiti dal server ad operazione ultimata.

Se la verifica della stato ha esito positivo (readyState=4 & status=200), il campo "value" dell'oggetto "Somma" viene aggiornato con il valore restituito dal server in **xmlhttp.responseText**.

Proviamo a comunicare con il nostro Web server SOMMA utilizzando il client AJAX realizzato. Aprire il file myForm.html con un browser web, inserire i numeri nei campi A e B e premere il pulsante "calcola". In figura 4.1 è riportato un esempio.

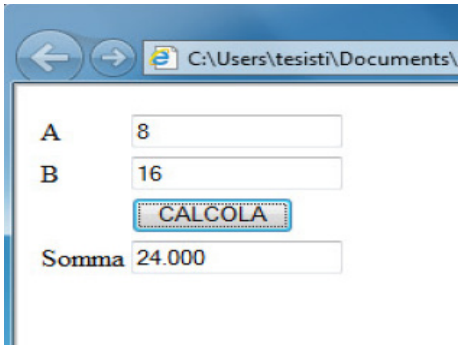


Figura 4.1: esempio di form HTML per la somma

4.2 Client per la misura con multimetro

Quello che presenteremo tra breve, è la realizzazione di un'interfaccia, lato client, che consenta all'utente di effettuare da remoto una misura con multimetro digitale.

La cosa a prima vista sembra complicata ma ci accorgeremo che non è così. Tutto quello che abbiamo presentato nel paragrafo precedente ci tornerà buono in questo lavoro, l'unica cosa da fare è rivedere un pò il codice HTML per la realizzazione del nostro form.

Il form, chiamato **myForm2**, sarà composto da 3 oggetti: un oggetto di tipo "menú a tendina" con il quale selezionare il tipo di misura (resistenza, tensioneDC, tensioneAC, ecc), un oggetto di tipo "input" di testo per visualizzare il risultato di misura, ed infine un oggetto di tipo "button" per inviare il comando. Per quanto riguarda il codice JavaScript, tutto rimane come prima.

Listing 4.4: creazione del "menú a tendina" per myForm2

```
<table width="350">
<tr>
  <td align="center">TIPO DI MISURA</td>
  <td><select name='a'>
    <option value='0'>resistenza</option>
    <option value='1'>DC voltage</option>
    <option value='2'>AC voltage</option>
    <option value='3'>DC current</option>
    <option value='4'>AC current</option>
  </select></td>
</tr>
```

All'oggetto "menú a tendina" viene associato un nome "a", e un valore che cambia a seconda della misura scelta. Ad esempio selezionando la casella "resistenza" avremo **a=0**.

L'utente che accede al Web service per la misura con multimetro, utilizzando come client myForm2, potrà scegliere il tipo di misura selezionando dal menú proposto, senza preoccuparsi di quale valore deve essere fornito al server. Premendo sul pulsante "calcola" sarà inviata la stringa HTTP nella forma:

http://127.0.0.1:8080/project/multimetro?a=valore

Aprire il form myForm2.html con un web browser, scegliere la misura selezionado dal menù a tendina e inviare la richiesta al server digitando sul pulsante MISURA. In figura 4.2 è riportato un esempio del form realizzato.

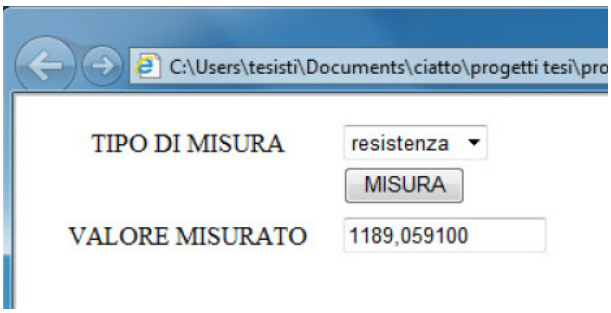


Figura 4.2: esempio di misura da multimetro con form HTML

CAPITOLO

5

JAVAFX

5.1 Introduzione a JavaFX

Da Sun e dal mondo Java, spesso, sono arrivate quelle innovazioni tecnologiche, poi riprese e rielaborate da altri, che hanno posto le basi per l'evoluzione dei sistemi informatici e del Web. Basti pensare alle applet e all'idea di virtual machine: un runtime engine che permette di eseguire lo stesso codice su qualunque browser e su qualunque macchina.

Quelle erano le basi per le attuali RIA, idea cavalcata poi da Macromedia (Adobe) e che oggi si declina anche in chiave Microsoft e JavaScript. Sun però non è rimasta a guardare e mette a disposizione **JavaFX**.

JavaFX è una famiglia di software applicativi, basati sulla Piattaforma Java, per la creazione di rich Internet applications, applicazioni web che hanno tutte le caratteristiche e funzionalità delle comuni applicazioni per computer. Con JavaFX è possibile realizzare delle applicazioni per computer, cellulari, dispositivi portatili di vario genere, televisori e altri tipi di piattaforme.

JavaFX include, oltre ad una crescente libreria di funzionalità grafiche, un vero e proprio linguaggio di programmazione indipendente da Java, chiamato "**JavaFX script**", un linguaggio di scripting dichiarativo e staticamente tipizzato fortemente orientato alla programmazione grafica; questo rende la

realizzazione di rich Internet application, e di applicazioni grafiche in genere, particolarmente agevolata.

In aggiunta al pacchetto **JavaFX SDK**, che include compilatore, utilità per l'esecuzione, e tutta la libreria JavaFX necessaria per lo sviluppo, sono stati rilasciati alcuni strumenti che rendono lo sviluppo molto piú agile:

- Un componente aggiuntivo per **NetBeans** che integra tutte le fasi di sviluppo JavaFX in un unico IDE.
- JavaFX Production Suite: utilità varie per facilitare il passaggio da programmi di grafica (Adobe Photoshop e Adobe Illustrator) e da formati come SVG, a codice JavaFX.

La sintassi di JavaFX è più snella e descrittiva rispetto a Java. Il linguaggio di scripting, pienamente supportato dalla JRE (Java Runtime Environment), si chiama "JavaFX Script Programming Language" o semplicemente **JavaFX script**. Esaminiamone le caratteristiche principali.

5.2 Il linguaggio JavaFX script

Nei paragrafi che seguono introdurremo il linguaggio javaFX script. Lo scopo è quello di presentare ai lettori, che già conoscono Java, alcune regole sintattiche proprie di javaFX.

Tipi e Variabili

In JavaFX script, possiamo definire le variabili dichiarando il tipo di dato in maniera esplicita oppure effettuando una dichiarazione generica. In ogni caso per dichiarare variabili e costanti ci serviamo rispettivamente delle parole chiave var e def.

Usando la chiave **var** si definiscono variabili modificabili, mentre con **def** variabili di sola lettura.

Listing 5.1: variabili

```
1 def length = 100;        //sola lettura
2 var count = 0;           //lettura scrittura
3 count++;
4 length = 500;            //compile error
```

JavaFX è un linguaggio staticamente tipizzato con i seguenti tipi di variabili: Boolean, Integer, Number, String, Void e Duration. Boolean viene

utilizzato per definire variabili booleane, Integer per variabili intere, Number per variabili di tipo float, e String per le stringhe. Diversamente da Java non è prevista la differenza tra i tipi Char e String in quanto è utilizzato solo il tipo String.

Void è usato con funzioni che non ritornano valori mentre Duration definisce variabili temporali (es: 100ms, 12s, 5h).

Listing 5.2: tipi di variabili

```
var element: Boolean;   //tipo booleano, false di default
var flag = true;
var counter: Integer;   //tipo intero (32bits), 0 di default
var radius: Number;     //tipo Number, 0.0 di default
var value = 1.1;
var s: String;          // "" di default
var s1 = 'duck';
var s2 = 'soup';
var s2 = "{s1}{s2}";    //"duck soup"
var s4 = s1 s2;         //compile error
var s5 = "good" "lucky"; //"goodlucky"
```

Classi e Oggetti

In questa sezione mostreremo come definire nuove classi e creare oggetti in javaFX. La classe Address di esempio definisce tre variabili di tipo String,

Listing 5.3: Adress class

```
public class Address {
 public var street: String;
 public var city: String;
 public var state: String;
}
//la classe può essere istanziata
//creando un oggetto di tipo "Address".
Address {
 street: "1 Main Street";
 city: "Santa Clara"; state: "CA";
}
```

chiamate variabili di istanza. Queste variabili sono sempre chiamate con

riferimento ad un oggetto specifico, vediamo come creare un oggetto da una classe.

Nei linguaggi di programmazione orientati agli oggetti, si utilizzano speciali funzioni chiamate **costruttori** per istanziare oggetti da una classe. In JavaFX è possibile costruire oggetti con espressioni definite **object literals**. La creazione dell'istanza può essere fatta impostando direttamente le proprietà dell'oggetto (es: "street", "city", "state").

E' anche possibile annidare un oggetto all'interno di un altro, si veda l'esempio riportato in listing 5.4 .

Listing 5.4: Customer class

```
1  public class Customer {
2       public var firstName: String;
3       public var lastName: String;
4       public var address: Address;
5       public function printAddress(){
6          println("Street: {address.street}");
7          println("City: {address.city}");
8          println("State: {address.state}");
9       }
10 }
11 //Istanziamo un oggetto Customer
12 var customer = Customer {
13       firstName: "John"; lastName: "Doe";
14       address: Address {
15           street: "1 Main Street";
16           city: "Santa Clara"; state: "CA";
17          }
18 }
```

Sequenze

Una sequenza è una lista ordinata di oggetti, analogo ad un array in un altro linguaggio. Le sequenze sono molto potenti in javaFX, permettono di immagazzinare ogni tipo di dato, inclusi oggetti generici. In questa sezione mostreremo come creare e accedere a sequenze di dati e oggetti.

Binding

Una delle maggiori caratteristiche di JavaFX è il **binding**.

Listing 5.5: Sequenze

```
def numbers = [2, 3, 5, 7, 14];
var names = ["Chiara", "Luca", "Serena"];
def numbers: Integer[] = [2, 3, 5, 7, 14];
var names: String[] = ["Chiara", "Luca", "Serena"];
var firstname = name[0];   //firstname è "Chiara"
var lastname = name[2];   //firstname è "Serena"

def w = [1..5];        //[1, 2, 3, 4, 5]
def h = [1.1..5.1];    //[1.1, 2.1, 3.1, 4.1, 5.1]
var j = [1..<5];       //[1, 2, 3, 4]
var k = [1..5 step 2];   //[1, 3, 5]

//creare sequenze con for
var step = for(n in [1..7 step 2]) n; //[1, 3, 5, 7]
var cube = for(n in [1..4]) n*n*n; //[1, 8, 27, 64]
```

L'idea consiste nella possibilità di associare ad una variabile una espressione. Ogni volta che l'espressione cambia, la variabile viene aggiornata automaticamente.

Questo semplifica estremamente la comune interazione tra componenti grafiche: per esempio si può facilmente associare il valore di un qualunque controllo, ad esempio una barra di scorrimento, all'attributo di un altro componente, come la dimensione di un'immagine, o la velocità di un'animazione. In listing 5.6 è riportato un esempio di binding.

Listing 5.6: Binding

```
var a = 10; var b = 2;
def product = bind a * b;
println(product);           //20
a = 100;
println(product);           //200
b = 5;
println(product);           //500
product = 10;               //compile error
```

Hello World in JavaFX

Di seguito illustriamo come realizzare una semplice applicazione, un "Hello World" di poco modificato, mostrando come vengono visualizzati in uscita i valori di un array di oggetti.

Il codice sorgente va salvato in un file con estensione .fx, da cui viene generato il .class, una volta compilato. La main-class è definita da un'istanza di javafx.stage.Stage, che rappresenta lo spazio in cui l'applicazione viene eseguita.

Listing 5.7: esempio di Hello World!

```
1  import javafx.stage.Stage;
2  import javafx.scene.Scene;
3  import javafx.scene.text.Text;
4  import javafx.scene.text.Font;
5
6  var numbers: Integer[] = [10..30 step 10];
7  insert 40 after numbers[30];
8  var hello: Text[]; var yAxis = 30;
9  for(n in [0..<sizeof(numbers)]){
10      hello[n] = Text {
11                  font : Font { size:numbers[n] }
12                  x: 10  y: yAxis
13                  content: "Hello World!" }
14              yAxis = yAxis + 50;
15         }
16 Stage {
17     title: "Hello World!"
18     width: 250
19     height: 250
20     scene: Scene { content: [ hello ] }
21 }
```

Esaminiamo alcune delle classi utilizzate nel codice:

- **Stage**: è l'oggetto principale dell'applicazione, equivalente al main.
- **Scene**: questa classe è un contenitore per gli oggetti di tipo Node. All'interno dell'applicazione possiamo utilizzare più "scene".
- **Node**: la superclasse comune a tutti gli elementi visualizzabili.
- **Text**: usato per definire un'area di testo.

- **Font**: usato per definire il font dell'oggetto Text che lo referenzia.

Lo spazio grafico di lavoro di JavaFX è un oggetto Stage, radice di ogni applicazione JavaFX, che può intercambiare oggetti Scene, contenitori di generici componenti grafici (Node).

A tutti gli effetti un oggetto Scene è un albero di nodi, in cui ogni nodo può essere sia un contenitore (che ad esempio specifica la disposizione bidimensionale dei suoi sotto componenti), sia un nodo grafico, cioè un **lightweight component** con una opportuna rappresentazione ed una serie di attributi che ne descrivono le proprietà (posizione, dimensioni, colorazione ecc.). Il concetto ricorda molto quello di DOM di una pagina HTML, in cui possono essere disposti opportuni tag con alcuni attributi.

Per concludere il nostro primo esempio di programma javaFX, in figura 5.1 riportiamo il layout prodotto da Hello World.fx

Figura 5.1: Layout di Hello World.fx

CAPITOLO

6

CREAZIONE DI CLIENT IN JAVAFX

6.1 Client per la somma di due numeri in JavaFX

In questo paragrafo impareremo come creare un client javaFX in grado di comunicare con un Web server LabVIEW per la somma di due numeri. Sfrutteremo le potenzialità proprie del linguaggio javaFX per ottimizzare la grafica dei nostri applicativi.

Il primo passo consiste nel creare un semplice form in javaFX, fatto questo vedremo in che modo comunicare con il server.

Il form è composto dai seguenti oggetti grafici:

- Tre oggetti di classe **"TextBox"** per l'inserimento dei parametri **a** e **b** e la visualizzazione del risultato.
- Un oggetto di classe **"Button"** per l'invio del comando al server.
- Oggetti di classe **"Text"** per personalizzare le stringe di testo (es: dimensioni, colorazione, ecc).

Da quanto appreso nel paragrafo 5.2, sappiamo che lo spazio grafico di lavoro di JavaFX, è un oggetto Stage, radice di ogni applicazione JavaFX, il quale al suo interno definisce oggetti Scene, contenitori di generici componenti grafici (Node).

Il contenuto del nostro oggetto Scene sarà un oggetto della classe **VBox** che chiameremo **form**.

VBox e **HBox** sono classi utilizzate per gestire il layout di sequenze di oggetti grafici. VBox ci consentirà di distribuire con un layout verticale gli oggetti TextBox e Button per creare il nostro form. La variabile form è quindi un oggetto di classe VBox, che separa verticalmente oggetti diversi. Ogni oggetto può essere a sua volta contenitore di nuovi oggetti così da formare un albero di nodi. Dal codice riportato in listing 6.1, per la creazione del form somma, si vede come VBox definisce al suo interno nodi di classe HBox.

Listing 6.1: form per la somma

```
1  var a_input=TextBox{text:"0" columns:12}
2  var b_input=TextBox{text:"0" columns:12}
3  var s_output=TextBox{text:"0" columns:12}
4  var ok_button=Button{ text:"Calcola"
5                action:function(){getData()}}
6  var form: VBox = VBox{
7    spacing: 10 //inserisce 10 pixel tra ogni elemento
8    nodeHPos: HPos.CENTER
9    content:[
10    Label { text: "Somma di due numeri" },
11    Separator {},
12    HBox{spacing:30 content:[Label{text:"A"},a_input]}
13    HBox{spacing:30 content:[Label{text:"B"},b_input]}
14    Separator {},
15    ok_button,
16    Separator {},
17    HBox{content:[Label{text:"Somma"},s_output]}
18    ]//content
19    }//form
```

La classe **"Button"**, possiede una propietà chiamata **action** che invoca una funzione quando l'utente preme il pulsante denominato calcola. La funzione chiamata invia una richiesta HTTP al server.

Al fine di dialogare con il server è necessario creare un oggetto di classe **HttpRequest** definita nel pacchetto **javafx.io.http**. HttpRequest permette

di specificare una location di destinazione (URL), un metodo (come GET o POST) e di inviare la richiesta HTTP tramite la funzione **start()**.

Alcune sue proprietà quali: **started, connecting, writing, reading e done**, possono cambiare stato nel corso di un'operazione, altre come: **onStarted, onConnection, onRead, onInput e onException**, richiamano specifiche funzioni. Per richieste con grande quantità di dati in arrivo, è possibile monitorare la percentuale di completamento dell'operazione utilizzando le proprietà **read e toread**.

A titolo di esempio vediamo in listing 6.2 come istanziare un oggetto di classe HttpRequest per inviare una richiesta HTTP al server LabVIEW per la somma di due numeri.

Listing 6.2: Oggetto HttpRequest

```
1  var http = HttpRequest{
2   location: "{REST}?{createArgList(data)}";
3   method: HttpRequest.GET;
4   onConnecting:function(){println("onConnecting")}
5   onDoneConnect:function(){println("onDoneConnect")}
6   onResponseHeaders:function(headerNames:String[]){
7   println("{headerNames.size()} response headers:");
8    for( k in headerNames ){
9  println("{k}:{getRequest.getResponseHeaderValue(k)}");
10     }
11   onInput:function(ip: InputStream){
12     var x: Integer;
13     var s1: String;
14     var s2: String;
15      while(true){
16       x = ip.read();
17       if(x == -1 or x == 46) break;
18       s2=Integer.toString(x-48);
19       s1="{s1}{s2}";
20       }
21       s_output.text = s1;
22    }
23   onDone:function(){println("onDone")}
24   }
25   http.start(); //invio comando http al server
```

La proprietà **onInput** della classe HttpRequest, definisce una funzione in grado di leggere i dati in arrivo dal server. La funzione per come è definita,

non restituisce valori e riceve come parametro di ingresso un oggetto di classe **InputStream**.

I programmi Java comunicano con le interfaccie I/O mediante flussi (stream), che sono sequenze ordinate di dati. Tutti i flussi si comportano allo stesso modo, a prescindere dal dispositivo fisico al quale sono collegati, le stesse classi e gli stessi metodi di I/O possono essere applicati a qualunque tipo di dispositivo (console, file, connessione di rete). Java implementa i flussi all'interno di una gerarchia di classi definite nel pacchetto **java.io**.

La classe astratta **InputStream** dichiara i metodi per leggere flussi binari da una sorgente specifica. Vediamo alcuni metodi:

- public abstract int **read()** throws IOException
- public void **close()** throws IOException

Il metodo **read()** legge un singolo byte dal flusso di input e lo restituisce come valore intero, restituendo -1 quando viene raggiunta la fine del flusso. Alla fine bisogna chiudere il flusso di input con il metodo **close()**.

Per i nostri scopi, è stato utilizzato il metodo **read()** per leggere i byte in arrivo dal server. Particolare attenzione deve essere prestata alla formattazione corretta dei dati (ogni singolo byte arriva con codifica ASCII).

Concludiamo il paragrafo riassumendo i blocchi principali che compongono il nostro client javaFX:

- Il form è un oggetto di classe VBox che separa verticalmente i seguenti oggetti grafici:
  - **TextBox**: crea un area di testo editabile. Noi la utilizzeremo per inserire i valori per i parametri **a** e **b** e visualizzare il risultato **somma**.
  - **Button**: Se si verifica un evento specifico, invoca una funzione per l'invio di una richiesta HTTP.
  - **Text**: possiede delle proprietà per personalizzare una stringa alfanumerica (es: dimensioni, colorazione, tipo di caratteri, ecc).
  - **HBox**: per disporre con un layout orizzontale oggetti diversi.
- Quando l'utente preme il bottone, viene invocata la funzione **getData()**. Questa funzione invia una richiesta HTTP al server, e per farlo definisce un oggetto di classe HttpRequest.
- Il flusso dati di risposta del server, viene acquisito definendo una funzione per la propietà **onInput** di HttpRequest. La funzione riceve un

oggetto di classe InputStream e sfrutta il metodo **read()** per la lettura dei byte in arrivo.

Per completezza di analisi riportiamo in figura 6.1 come si presenta il nostro client javaFX per la somma di due numeri.

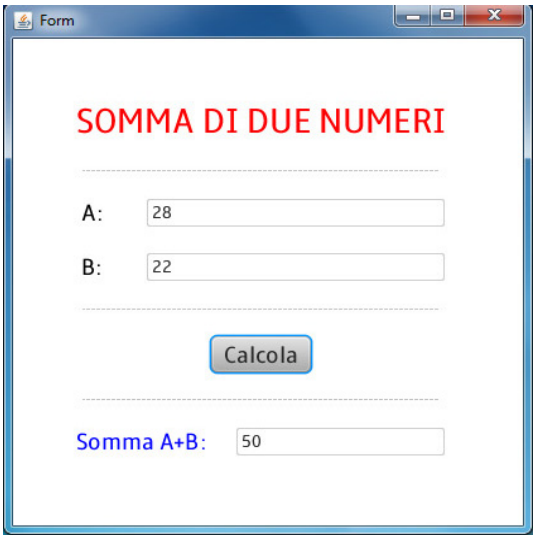


Figura 6.1: client per la somma di due numeri

6.2 Client misura con multimetro in JavaFX

In questo paragrafo svilupperemo un client javaFX per realizzare da remoto delle misurazioni su un multimetro digitale. Quanto presentato nel paragrafo precedente sarà indispensabile anche in questo lavoro, basterà modificare leggermente il codice per la creazione del nostro form.

Il nostro client javaFX dovrà presentare un'interfaccia grafica che consenta di:

- Scegliere il tipo di misura da fare con il multimetro digitale (es: resistenza, tensioneDC, tensioneAC, ecc).
- Digitando sul pulsante "misura", inviare una richiesta ad un server remoto sfruttando il protocollo HTTP.
- Visualizzare il risultato della misura.

Per lo sviluppo è necessario ricordare che il nostro server LabVIEW selezionerà la misura con il seguente criterio:

- **0** resistenza
- **1** tensioneDC
- **2** tensioneAC
- **3** correnteDC
- **4** correnteAC

Utilizzeremo quindi un oggetto di classe **SwingComboBox**. Questa classe consente di creare un "menú a tendina" nel quale è possibile associare un valore numerico ad ogni opzione (es: associare valore=0 all'opzione "resistenza").

Selezionato il tipo di misura dal menú proposto, il valore corrispondente all'opzione scelta sarà "appeso" come parametro per la stringa HTTP (metodo GET). Il server imposterà il multimetro in base al valore del parametro.

Il flusso di dati in arrivo dal server, come per la somma, verrà aquisito sfruttando il metodo read() della classe InputStream. Rispetto al caso precedente però bisognerà fare attenzione alla presenza del punto (a rappresentare la virgola) nelle misure effettuate (codice ASCII 46).

In listing 6.3 vediamo un frammento del codice per la proprietà **onInput**.

Listing 6.3: frammento di codice della proprietà onInput

```
1 while(true){
2  x = ip.read();
3  if(x == -1) break;
4  a = x-48;
5  if(a >= 0){ s2=Integer.toString(a); }
6  else{ s2=","; }
7  s1="{s1}{s2}";
8 }
```

Concludiamo riportando in figura 6.2 come si presenta il nostro client javaFX per la misura con multimetro. L'esempio propone una misura su una reistenza.

6.3 Client misure ripetute con multimetro in JavaFX

Nei paragrafi precedenti abbiamo visto come realizzare applicativi javaFX capaci di comunicare con un server remoto utilizzando il protocollo HTTP.

Figura 6.2: client per la misura con multimetro digitale

In questa sezione realizzeremo un client javaFX in grado di effettuare da remoto **misure ripetute** su multimetro digitale, visualizzando in un grafico i valori ottenuti, e calcolando media e incertezza standard sulle **n** misure.

L'utente disporrà di un interfaccia grafica che consenta di:

- Scegliere il tipo di misura da effettuare sul multimetro digitale, selezionando da un "menú a tendina".
- Impostare un numero **n** di misure ripetute.
- Inviare la richiesta al server remoto digitando sul pulsante "misura". Si è scelto di utilizzare un oggetto di tipo "Button", per lasciare la possibilità all'utente di controllare i parametri impostati prima di spedirli.
- Visualizzare un grafico con i risultati delle misure.
- Visualizzare valore medio e incertezza standard sulle **n** misure.

JavaFX consente di gestire sei tipologie di grafici, per lo piú bidimensionali, che sono: Area Chart, Bar Chart, Bubble Chart, Line Chart, Pie Chart e Scatter Chart, di cui cinque hanno un antenato comune, la classe XYChart che discende a sua volta dalla superclasse **Chart**.

Esistono quindi elementi comuni alle diverse tipologie di grafici, proprie dalla superclasse **Chart**, come la legenda, il titolo, e alcune proprietà come

lo spazio tra area del grafico e titolo, i tipi di caratteri, gli sfondi e altro ancora.

Il nostro nuovo form sarà composto da un oggetto di classe **LineChart**. Questa classe ci permetterà di realizzare un grafico bidimensionale per visualizzare le misure effettuate.

La creazione dell'istanza può essere fatta impostando direttamente le proprietà dell'oggetto, come **data, xAxis** e **yAxis**.

La proprietà **data** sarà un oggetto della classe **LineChart.Series** che al suo interno contiene una sequenza di oggetti **LineChart.Data**. Ogni oggetto LineChart.Data, definisce tramite le proprietà **xValue** e **yValue**, le coordinate di un punto nel grafico. Quindi, ogni coppia di coordinate è un oggetto di classe LineChart.Data.

Le proprietà **xAxis** e **yAxis** sono necessarie per definire caratteristiche degli assi, quali lowerBound, upperBound, tickUnit, ecc. JavaFX dispone di un pacchetto, **javafx.scene.chart.part** che include tutta una serie di classi utilizzabili per personalizzare gli assi del grafico. Nel nostro caso xAxis e yAxis sono definite da oggetti della classe **NumberAxis**.

Riportiamo in listing 6.4 un esempio di come realizzare un semplice grafico bidimensionale sfruttando la classe **LineChart**.

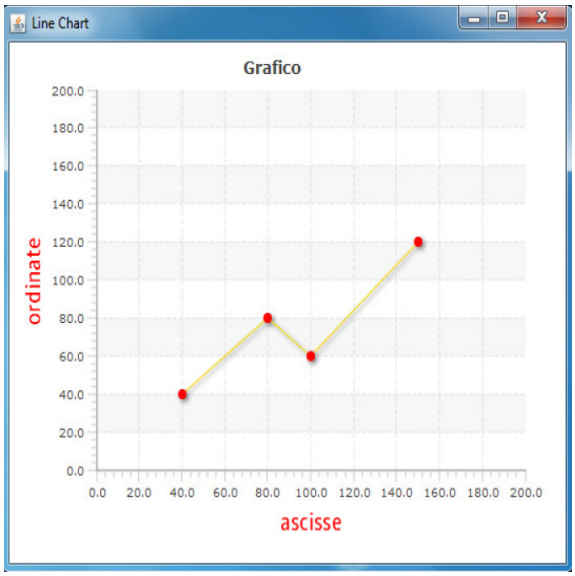


Figura 6.3: Grafico ottenuto con la classe LineChart

Visto come gestire grafici bidimensionali in javaFX, passiamo oltre e analizziamo in che modo formattare correttamente il flusso dati in arrivo dal server.

Listing 6.4: Esempio di grafico con LineChart

```
def lineChart = LineChart {
 title: "Grafico"
 xAxis: NumberAxis{
 label:"ascisse"
 labelFill:Color.RED labelFont:Font{size: 20}
 upperBound:200  tickUnit:20  minorTickCount:5
 }
 yAxis: NumberAxis{
 label:"ordinate"
 labelFill:Color.RED labelFont:Font{size: 20}
 upperBound:200  tickUnit:20 minorTickCount:5
 }
 data:[ LineChart.Series{
  data:[
  LineChart.Data{xValue:40 yValue:40 fill:Color.RED}
  LineChart.Data{xValue:80 yValue:80 fill:Color.RED}
  LineChart.Data{xValue:100 yValue:60 fill:Color.RED}
  LineChart.Data{xValue:150 yValue:120 fill:Color.RED}
  ]
 }]
}
```

I dati in arrivo dal server dovranno essere convertiti in valore numerico (tipo Float) e salvati in una sequenza (array) di numeri. Questa operazione è indispensabile per poter graficare le misure e riutilizzare i dati per successive operazioni (es: calcolo di media e incertezza standard)

Per poter recuperare, lato client, ogni singola misura dal flusso dati in arrivo, il server LabVIEW, interpone un carattere corrispondente ad uno spazio vuoto, dopo ogni valore misurato. A tal fine, il codice ASCII 32 (spazio vuoto), sarà interpretato come un "messaggio in codice" che indica la fine di una misura e l'inizio della successiva. Per completezza, in listing 6.5 è presente un frammento di codice per la proprietà **onInput**, necessario a formattare correttamente il flusso di dati in arrivo, e salvare ogni misura in una sequenza (array) di Float.

Recuperata ogni singola misura lato client, manca da vedere come calcolare media e incertezza standard sulle **n** misure. Per questo scopo faremo qualche richiamo teorico sull'argomento.

La misura di una grandezza è sempre affetta da **un'incertezza intrinseca**, corrispondente al grado di indeterminatezza associato al valore misurato.

Listing 6.5: proprietà onInput

```
onInput: function(ip: InputStream){
while(true){
 //read() legge ogni singolo byte dal flusso di input
 //restituendo -1 se si raggiunge la fine del flusso.
 x = ip.read();
 if(x == -1) break;
 //x-48 mi restituisce il numero decimale corretto.
 a = x-48;   //48 è il codice ASCII per lo zero.
 //se a>=0 convertiamo il numero intero in stringa
 //se a<0 non abbiamo un numero decimale.
 //abbiamo il punto (ASCII=46) oppure uno spazio vuoto.
 if(a >= 0){s2=Integer.toString(a);}
 //x==32 (ASCII per lo spazio) indica la fine di una misura.
 //salviamo quindi la misura in un array di Float.
 else if ( x==32){
 try{ h[i] = Float.parseFloat(s1);
 }catch(Exception){}
 //incrementiamo l'indice i per l'array di misure.
 //annullo s1 e s2 per la misura successiva.
 i=i+1;
 s1=""; s2="";
 }
 else  s2=".";
 s1="{s1}{s2}";
 } }
```

Misure ripetute dalla stessa quantità non forniscono mai lo stesso valore ma presentano una certa **dispersione attorno ad un valore medio**.

Il risultato di una misurazione è solo un'approssimazione (stima) del valore vero del misurando e quindi ha significato solo se fornito assieme alla sua **incertezza**.

La Guida all'Espressione dell'Incertezza di Misura (GUM), distingue due tipologie di incertezza in base alla modalità di valutazione:

- **TIPO A**: incertezze di misura valutabili attraverso **metodi statistici** applicati a osservazioni ripetute dello stesso misurando.

- **TIPO B**: incertezze valutabili con metodi non statistici, basati su conoscenze a priori (specifiche costruttore, dati di taratura, ecc).

Lo scopo di tale classificazione è indicare due possibili **modi operativi** con i quali valutare le componenti di incertezza. Entrambi i metodi di valutazione sono basati su distribuzioni di probabilità, e le componenti di incertezza vengono quantificate per mezzo di **varianze** e **deviazioni standard**.

La varianza u^2, che caratterizza una componente di tipo A, viene stimata da una serie di osservazioni ripetute. La radice quadrata della varianza, indicata con **u** e corrispondente alla deviazione standard, viene detta **incertezza standard di tipo A**.

Supponiamo di disporre di N realizzazioni INDIPENDENTI (osservazioni), indicate con x_i per i=1,2,...,N della v.a. X (misurando). Uno stimatore in grado di fornire una stima del valore del misurando si basa sul calcolo della **media campionaria**:

$$\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

Le singole osservazioni x_i differiscono dal valore medio a causa di fluttuazioni aleatorie. La **varianza campionaria** fornisce una stima della distribuzione delle singole osservazioni attorno al valore medio:

$$s^2(x) = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \overline{x})^2$$

La stima invece della varianza asscociata al valore medio (risultato di misura) è data da:

$$s^2(\overline{x}) = \frac{1}{N} s^2(x)$$

Quantifica il grado di approssimazione tra stima del valore medio e valore atteso del misurando. Tale varianza viene usata come misura dell'incertezza associata a X e viene detta incertezza standard di tipo A:

$$s^2(\overline{x}) = u_A^2(x)$$

Sebbene la quantità fondamentale sia la varianza, per motivi pratici si considera solitamente la deviazione standard $u_A(x)$.

Ricapitolando, riportiamo le formule utili ai nostri scopi, per il calcolo di media e incertezza standard:

1. **Media campionaria**: $\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$
2. **Incertezza standard**: $u_A(x) = s(\overline{x}) = \sqrt{\frac{1}{N(N-1)} \sum_{i=1}^{N} (x_i - \overline{x})^2}$

Listing 6.6: funzioni javaFX per media e incertezza standard

```
//l'array z[] contiene le misure con multimetro.
function incertezza_standard():String{
 var f: Float;  var N: Integer;
 for( j in [0..<b_input.value] ){
   f = f + (z[j] - media2())*(z[j] - media2());
   N++; }
    f = f/( N*(N-1) );
    f = Math.sqrt(f);
    return Float.toString(f)
}
//funzione per il calcolo della media campionaria
function media2(): Float{
 var f: Float = 0;
 var count: Integer = 0;
 for(j in [0..<b_input.value]){
   f = f + z[j];
   count++; }
  f = f/count;
  return f
}
```

Per il calcolo realizzeremo delle funzioni javaFX. A titolo di esempio, in listing 6.6 è presente il codice per media e incertezza standard sulle N misure:

Arrivati a questo punto abbiamo tutto il necessario per sviluppare il lavoro. Riassumiamo di seguito le parti principali che compongono il nostro client javaFX.

1. Il form è un oggetto di classe VBox che separa verticalmente i seguenti oggetti grafici:

   - **SwingComboBox**: per la creazione del "menú a tendina". E possibile associare un valore numerico ad ogni etichetta (es: text: "resistenza" value: 0).
   - **SwingSlider**: associa un range di valori ad una barra a scorrimento.
   - **Button**: Se si verifica un evento specifico, invoca una funzione per l'invio di una richiesta HTTP.
   - **LineChart**: per la creazione di un grafico bidimensionale.

- **TextBox**: crea un area di testo editabile. Noi la utilizzeremo per visualizzare i risultati di media e incertezza standard.
- **Text**: possiede delle proprietà per personalizzare una stringa alfanumerica (es: dimensioni, colorazione, tipo di caratteri, ecc).
- **HBox**: per disporre con un layout orizzontale oggetti diversi.

2. Quando l'utente preme il bottone, viene invocata la funzione **getData()**. Questa funzione invia una richiesta HTTP al server, e per farlo definisce un oggetto di classe HttpRequest.

3. Il flusso dati di risposta del server, che è una query con tutte le **n** misure effettuate, viene convertito e salvato in una sequenza (array) di tipo Float. L'array consente di:

   - Aggiornare le variabili xValue e yValue dell'oggetto LineChart.Data.
   - Calcolare valore massimo e minimo tra le n misure in modo da impostare un range per l'asse delle ordinate, tramite le proprietà lowerBound e upperBound.
   - Calcolare media e incertezza standard sulle **n** misure.

4. Per aggiornare dinamicamente le proprietà del grafico vengono definite delle variabili con la clausola **bind**.

Concludiamo il paragrafo riportando in figura 6.4 come si presenta il nostro client javaFX. L'esempio propone 10 misure ripetute su resistenza.

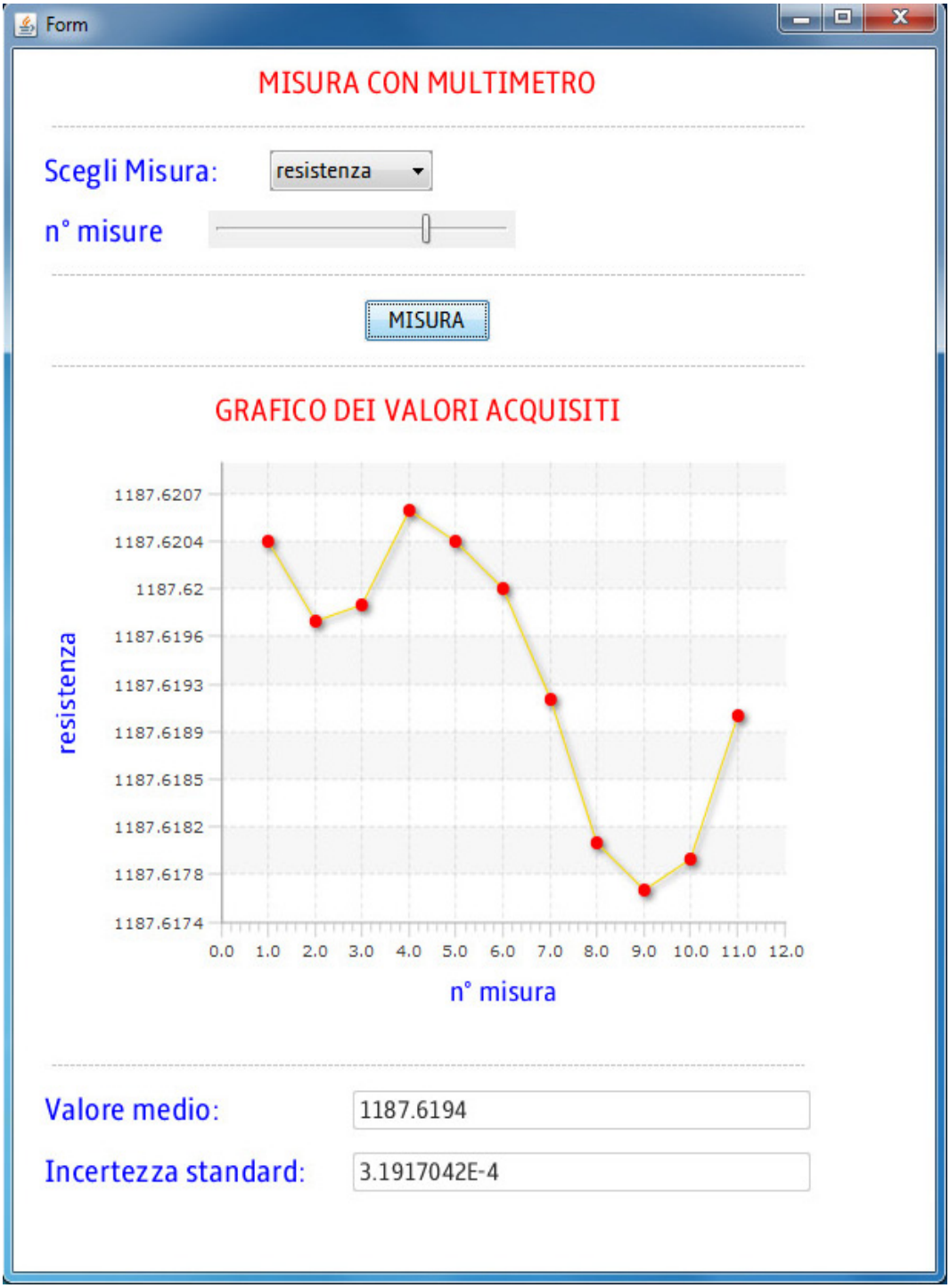


Figura 6.4: Client misure ripetute con multimetro digitale in javaFX

CAPITOLO

7

GESTIONE DI PIÙ CLIENT

In questo capitolo vedremo come fare per tenere in memoria lato server le informazioni relative agli utenti che accedono al servizio. Prima di cominciare spieghiamo due concetti importanti, i cookie e le sessioni.

7.1 Cookie

I cookie HTTP (più comunemente denominati web cookies, o semplicemente cookie) sono frammenti di testo inviati da un server ad un web client (di solito un browser), e poi rimandati indietro dal client al server senza subire modifiche, ogni volta che il client accede allo stesso server.

Sono usati per eseguire autenticazioni e tracking di sessioni e memorizzare informazioni specifiche riguardanti gli utenti che accedono al server, come ad esempio i siti preferiti o, in caso di acquisti on-line, il contenuto dei loro "carrelli della spesa" (shopping cart).

I cookie vengono spesso erroneamente ritenuti veri e propri programmi. In realtà essi sono semplici blocchi di dati, incapaci, da soli, di compiere qualsiasi azione sul computer. In particolare non possono essere né spyware, né virus. I moderni browser permettono agli utenti di decidere se accettare o no i cookie, e l'eventuale rifiuto rende alcuni oggetti inutilizzabili. Ad

esempio, gli shopping cart implementati con i cookie non funzionano in caso di rifiuto.

Vediamo alcuni dei diversi utilizzi:

- Per riempire il carrello della spesa virtuale in siti commerciali (i cookie ci permettono di mettere o togliere gli articoli dal carrello in qualsiasi momento).
- Per permettere ad un utente il login in un sito web.
- Per tracciare i percorsi dell'utente (tipicamente usato dalle compagnie pubblicitarie per ottenere informazioni sul navigatore, i suoi gusti le sue preferenze. Questi dati vengono usati per tracciare un profilo del visitatore in modo da presentare solo i banner pubblicitari che gli potrebbero interessare).
- Per la gestione di un sito: i cookie servono a chi si occupa dell'aggiornamento di un sito per capire in che modo avviene la visita degli utenti, quale percorso compiono all'interno del sito.

Un cookie, piu in dettaglio, è un **header** aggiuntivo presente in una richiesta (Cookie:) o risposta (Set-cookie:) HTTP: nel caso il server voglia assegnare un cookie all'utente, lo aggiungerà tra gli header di risposta. Il client deve notarne la presenza e memorizzarlo in un'area apposita (in genere, si usa una directory dove ogni cookie viene memorizzato in un file).

Il cookie è composto da una stringa di testo arbitraria, una data di scadenza (oltre la quale non deve essere considerato valido) e un pattern per riconoscere i domini a cui rimandarlo. Il browser client lo rimanderà senza alcuna modifica, allegandolo a tutte le richieste HTTP che soddisfano il pattern, entro la data di scadenza. Il server può quindi scegliere di assegnare il cookie di nuovo, sovrascrivendo quello vecchio.

Contrariamente a quanto comunemente si crede, un cookie non è un piccolo file di testo: può essere sì memorizzato in un file di testo, ma non necessariamente. Nel cookie solitamente possiamo trovare sei attributi:

- **Nome/valore** è una variabile ed un campo obbligatorio.
- **Dominio** (domain) ci permette di specificare il dominio di provenienza del cookie.
- **Scadenza** (expiration date) è un attributo opzionale che permette di stabilire la data di scadenza del cookie.

- **Percorso** (path) specifica il percorso dal quale il cookie viene mandato all'utente finale.

- **Modalità d'accesso** (HttpOnly) rende il cookie invisibile a javascript e altri linguaggi client-side presenti nella pagina.

- **Sicuro** (secure) indica se il cookie debba essere trasmesso criptato con HTTPS.

7.2 Sessioni

Una corretta gestione delle sessioni è determinante nello sviluppo di applicazioni web ad elevato grado di interazione con l'utente. Come sappiamo, infatti, il web non ha memoria o meglio ce l'ha grazie alle sessioni (e ai cookie).

La sessione è, in sintesi, un file di testo creato dal nostro sito e depositato in una cartella predefinita del server che ci ospita. La funzione di questo file di testo è quella di fungere da contenitore per diverse informazioni sull'utente cui è associato.

Questo file di testo viene associato ad un dato utente attraverso un cookie oppure con un richiamo nelle url navigate. Nel primo caso viene salvato sul PC dell'utente un cookie contenente l'ID della sessione (la scadenza di questo cookie è impostata di default alla chiusura del browser), nel secondo caso (generalmente utilizzato quando il client non accetta i cookie) l'ID della sessione viene passato in un parametro della URL sotto forma di querystring.

Quindi, il meccanismo delle sessioni permette di memorizzare informazioni sul cliente in maniera simile a come avviene per i cookie con le seguenti particolarità:

- Le informazioni importanti sono memorizzate nel server e non nel client.

- Si riducono le informazioni scambiate fra client e server, quindi si riduce il traffico in rete con benefici per prestazioni e sicurezza.

- Per associare al client la sessione, si invia un cookie leggero che contiene l'ID di sessione.

7.3 Web service misure con multimetro e gestione delle sessioni

In questo paragrafo presenteremo come realizzare un Web service in LabVIEW capace di gestire le sessioni utente, in particolare, amplieremo il nostro Web service, denominato: "misure ripetute con multimetro", in modo da tenere traccia degli utenti che accedono al servizio. Il progetto sarà strutturato come segue:

- Quando un utente accede al nostro server LabVIEW, il programma associerà al cliente un ID di sessione che lo identifichi ogni qualvolta riaccede al servizio.
- Oltre all'ID di sessione, il server terrà traccia di alcuni parametri, in particolare, il tipo di misura scelta (resistenza, tensioneDC, ecc) e il numero **n** di misure effettuate.
- In questo modo si riducono le informazioni scambiate tra client e server. Non è piu necessario inviare tutti i parametri ogni qualvolta si accede al servizio; questi saranno spediti solo nelle operazioni di setup, per cambiare le impostazioni di misura.
- Il server invierà al client, oltre al flusso di dati richiesto, un cookie leggero contenente l'ID della sessione, allegandolo all'header di risposta. Piu in dettaglio, riceveremo dal server un cookie contenente un nome (cookie:) e un valore (ID della sessione). Il valore corrisponde ad una stringa alfanumerica.
- Il software client salverà il cookie in una variabile, e ogni qualvolta l'utente riaccede al servizio, lo allegherà all'header della richiesta HTTP.

Riportiamo nelle figure 7.2 e 7.3 i VI LabVIEW per la realizzazione del nostro Web service; di seguito presentiamo i blocchi utilizzati per la gestione delle sessioni utente:

- Il VI ha un solo terminale associato all'unico controllo **httpRequestID**. Questo controllo deve essere connesso ad uno dei pin del connettore.
- **Check if session exist**: verifica se un ID di sessione è associato alla corrente richiesta HTTP. Il terminale di uscita (**session exist?**), restituisce una variabile booleana con valore true se la sessione esiste, false

se non esiste. Il valore ottenuto selezionerà una tra due altenative di una struttura case, in particolare:

1. **True**: il server recupera le variabili della sessione utente in questione, esegue le misure richieste e ritorna i dati al cliente.
2. **False**: Il server genera un session ID cookie per il nuovo utente, salva i parametri **a** e **b** ricevuti come variabili di sessione, esegue le misure, e infine invia dati e cookie al client.

- **Create session**: questo blocco associa alla richiesta HTTP un identificativo di sessione, sotto forma di cookie da inviare al client. Il cookie permette al server di riutilizzare la stessa sessione HTTP quando il medesimo client inoltra un'altra richiesta al server.
- **Set HTTP header**: associa un nome (cookie nel nostro esempio) all'ID di sessione creato. Quindi nell'header della nostra risposta HTTP, avremo un campo con nome (cookie) e valore (ID della sessione).
- **Read form data**: legge e ritorna i parametri **a** e **b** specificati nella stringa HTTP.
- **Write single session variable**: crea o sovrascrive una variabile di sessione. Nel nostro esempio il valore associato ai parametri **a** e **b** viene tenuto in memoria come variabili di sessione.
- **Read session variable**: legge e ritorna il valore della variabile di sessione specificata.

Visti i blocchi utilizzati per la gestione delle sessioni manca da analizzare la seconda parte del nostro programma, quella necessaria alla comunicazione con multimetro. A tal fine, tutto resta come già spiegato per la creazione del Web service misure ripetute con multimetro.

Il programma LabVIEW è realizzato allo stesso modo, anche se da figura 7.1 potrebbe sembrare diverso. Quello che abbiamo fatto è creare un subVI per rendere il codice piu leggibile. Il blocco presente nella nuova struttura case, è strutturato al suo interno come in figura 7.1 e riporta quanto analizzato nel paragrafo 3.5.4.

7.4 Client misure con multimetro e gestione delle sessioni

In questa sezione, analizzeremo le modifiche da apportare, al nostro client javaFX per misure ripetute con multimetro, in modo da svolgere alcune funzio-

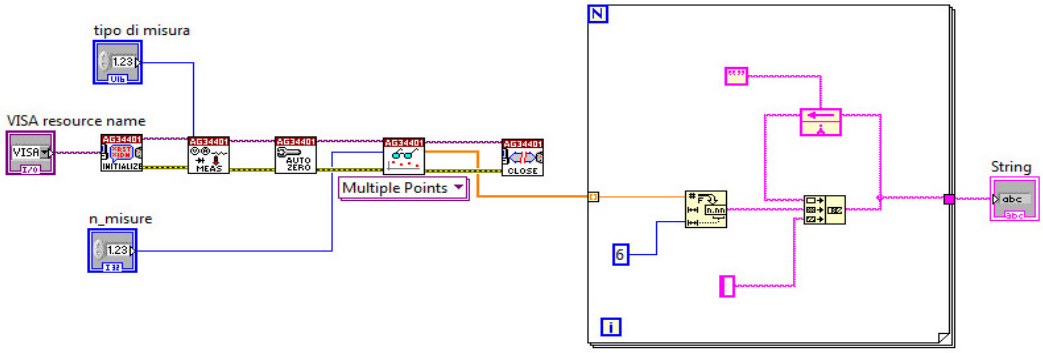


Figura 7.1: subVI per misure ripetute con multimetro

ni aggiuntive che ci consentano di sfruttare l'utilizzo delle sessioni lato server. In particolare, il nostro software client dovrà avere le seguenti caratteristiche:

1. Salvare in una variabile di programma il session ID cookie ricevuto dal server.

2. Ogni qualvolta l'utente riaccede al servizio, nei campi dell'header della richiesta HTTP dovrà aggiungere un campo contenente il cookie che lo identifica. In questo modo il server riconoscerà l'utente ad ogni suo accesso.

3. L'interfaccia grafica del nostro applicativo, dovrà disporre di due oggetti "button" diversi, per inviare una richiesta di misura al server:

   - **SETUP**: con questa opzione verrà inviata una richiesta HTTP al server allegando i parametri **a** e **b** per le impostazioni di misura.
   - **MISURA**: il client invierà un richiesta HTTP senza parametri aggiuntivi. Vengono quindi mantenute le impostazioni di setup precedentemente impostate. Questo consente di alleggerire la comunicazione con guadagni in termini di banda.

Per ricevere l'header di risposta del server, la classe HttpRequest mette a disposizione una proprietà chiamata **onResponseHeaders**. Questa definisce una funzione che riceve in ingresso un array di strighe, più precisamente, riceve un array contenete tutti i valori contenuti nell'header di risposta del server. Riportiamo in listing 7.1 il codice per la proprietà onResponseHeaders necessario a salvare in un array di strighe i campi value dell'header di risposta.

Listing 7.1: proprieta onResponseHerders

```
onResponseHeaders:function(headerNames: String[]){
println("{headerNames.size()} response headers:");
 for (name in headerNames){
  println("{name}:{http.getResponseHeaderValue(name)}");}
 for( j in [0..<headerNames.size()]){
valori_header[j]=http.getResponseHeaderValue(headerNames[j])
} }
```

Ogni qualvolta l'utente riaccede al servizio invierà il cookie sfruttando una proprietà della classe HttpRequest chiamata **headers**. Questa contiene una sequenza di oggetti **HttpHeader**, dove ogni oggetto definisce un campo dell'header. Sfruttando gli attributi **name** e **value** si imposta il campo di header richiesto. Riportiamo in listing 7.2 come definire l'header per l'invio del cookie.

Listing 7.2: proprieta headers

```
headers:[
  HttpHeader {
      name: "Cookie:";
      value: "set-cookie:{valori_header[8]}";
      }
]
```

Concludiamo il paragrafo riportando in figura 7.4 come si presenta il nostro client javaFX. L'esempio propone 20 misure ripetute su resistenza.

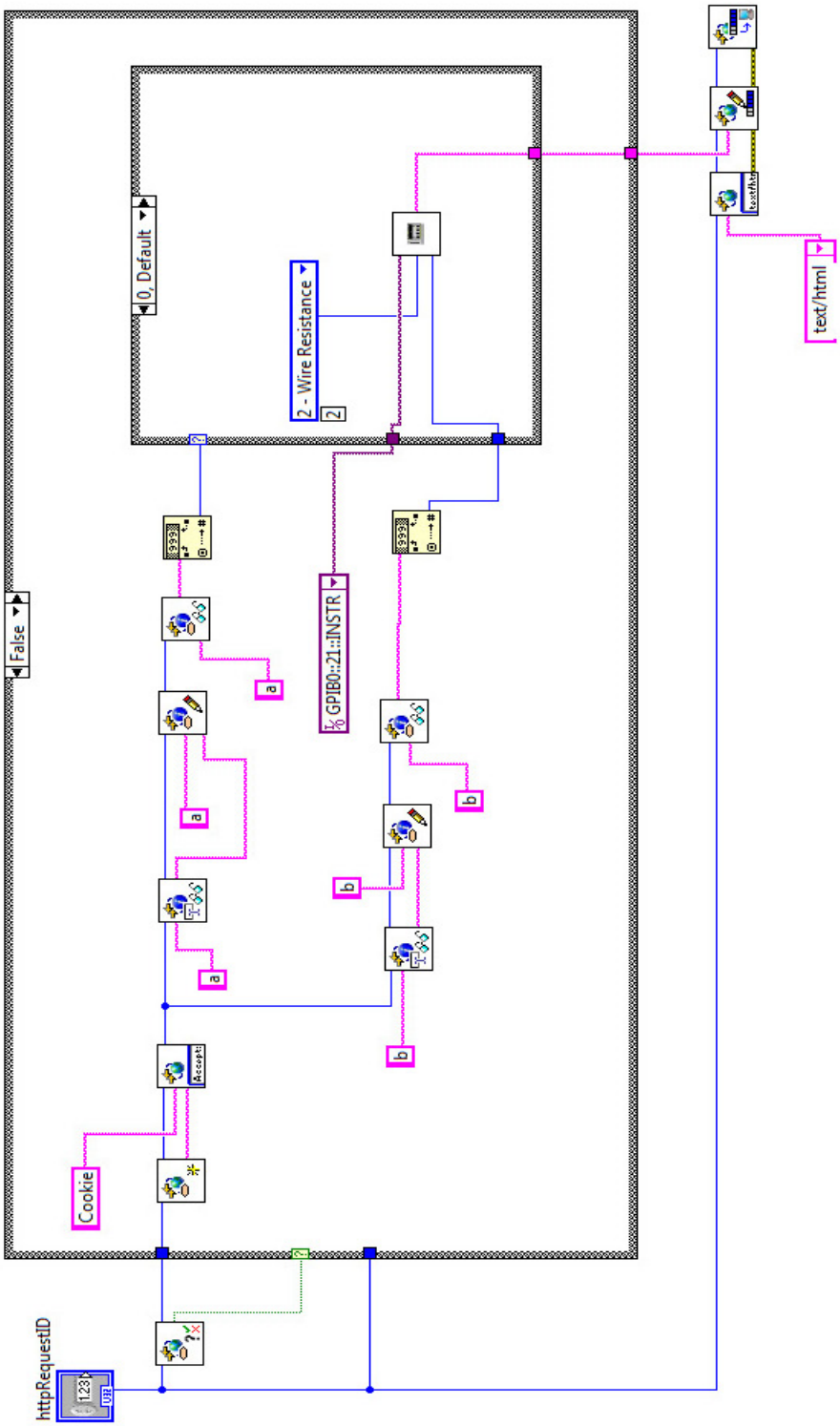


Figura 7.2: VI per la gestione delle sessioni (case: FALSE)

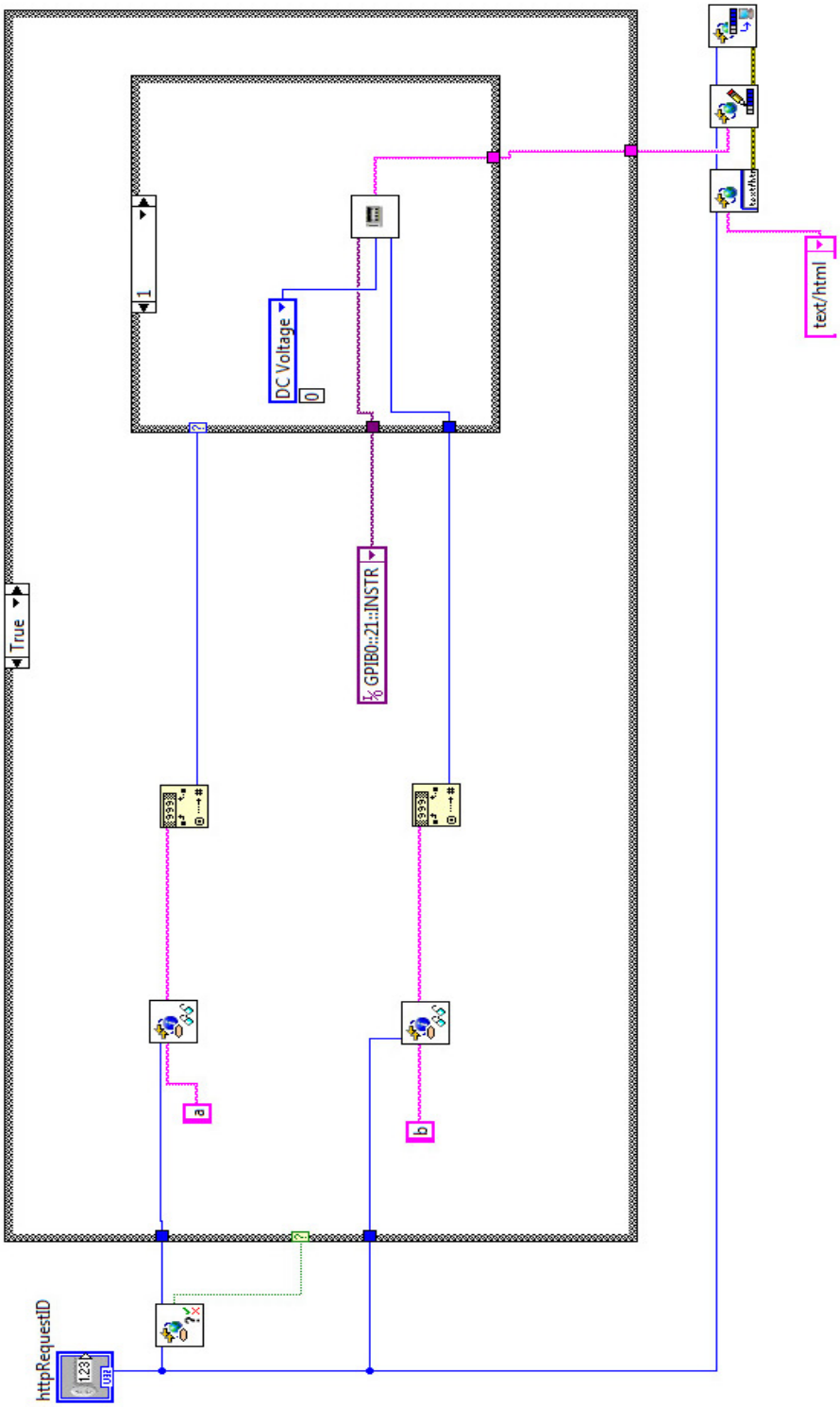


Figura 7.3: VI per la gestione delle sessioni (case: TRUE)

Figura 7.4: client javaFX per misure ripetute con multimetro

CAPITOLO

8

CONCLUSIONI

In questo capitolo, concluderemo il lavoro presentando i motivi che ci hanno spinto all'utilizzo di LabVIEW e javaFX. In ultima analisi discuteremo i risultati ottenuti.

L'idea di sviluppare un laboratorio remoto per misure elettroniche, parte proprio dalle possibilità offerte da LabVIEW. Questo ambiente di sviluppo viene utilizzato principalmente con i seguenti scopi:

- Acquisizione dati e gestione della strumentazione.
- Creazione di interfacce utente dedicate.
- Creazione di test dedicati per il controllo di apparati.

Esattamenete quello che serve per il nostro progetto: un software lato server capace di gestire la strumentazione in modo efficiente.

Oltre a queste caratteristiche, le possibilità offerte da LabVIEW di realizzare **Web service**, aprono la strada per un progetto innovativo che ci consenta di:

- Sviluppare un'interfaccia utente sfruttando un client personalizzato javaFX.
- Comunicare con applicazioni LabVIEW da qualsiasi dispositivo capace di connettersi al web.

- Comunicare sfruttando il protocollo HTTP e non un protocollo proprietario.

Per quanto riguarda l'utilizzo di javaFX come software di sviluppo lato client, questa resta solo una delle possibili scelte. Esistono molte alternative valide a javaFX, ma diversi sono i punti a suo favore che la rendono una scelta vincente:

- Utilizza un linguaggio di scripting veloce e intuitivo.
- Possiede una vasta libreria di funzionalità grafiche messe a disposizione degli sviluppatori.
- È un linguaggio fortemente orientato alla programmazione grafica per lo sviluppo di rich internet application (RIA). Esattamente quello che ci serve per realizzare interfacce utente interattive con alta velocità d'esecuzione.
- Il linguaggio di scripting, javaFX script, è pienamente supportato dalla JRE (Java Runtime Environment).
- È un software libero (open source). Il pacchetto per lo sviluppo (javaFX SDK) è scaricabile gratuitamente dal sito di javaFX.

In questo momento JavaFX non è l'unico attore a competere nel mercato delle RIA, anzi, l'entrata vera e propria nella competizione è avvenuta in ritardo rispetto ai concorrenti. Coloro che si possono ritenere come veri e propri antagonisti di JavaFx sono:

- AJAX / XHTML / CSS
- Flex della Adobe
- Silverlight della Microsoft
- Google GWT

Lo stato attuale delle cose non mostra JavaFX come un reale concorrente delle principali tecnologie citate sopra. Infatti, se per le tecnologie concorrenti notiamo in rete numerosi software, al contrario, software realizzati con piattaforma JavaFX, ce ne sono ben pochi. I motivi sono vari e, per citarne alcuni, possiamo partire da Ajax. Questa piattaforma esiste già da tempo ed e largamente utilizzata, seppur c'è da dire che, attualmente, l'offerta e le funzionalità messe a disposizione da JavaFX, sono molto piu utili a costruire applicazioni complesse con maggior facilità rispetto ad Ajax.

Sicuramente il re incontrastato dei linguaggi web per quanto riguarda applicazioni multimediali è Adobe Flex, in quanto negli anni ha trovato campo fertile tra gli sviluppatori per la realizzazione di applicazioni web. Ad oggi tutti i browser supportano il Flash, seppur con l'installazione di un plugin aggiuntivo. A guidare parte del processo di adozione di questa tecnologia da parte dei programmatori è stata senz'altro l'interpiattaforma, ossia la capacità di girare su piu ambienti diversi. Come esempio di adozione di questa tecnologia basta pensare a Youtube, il famoso portale di video sharing dove tutti i contenuti sono in Flash.

In ultima analisi, dai risultati ottenuti, riportiamo le nostre considerazioni finali:

- Le potenzialità offerte da LabVIEW per la gestione della strumentazione e la semplicità con cui permette lo sviluppo di Web service lo rendono un ideale software di sviluppo lato server.

- Lato client abbiamo molte alternative di sviluppo, ma i risultati ottenuti utilizzando javaFX sono stati ottimali. In aggiunta, la possibiltà di usufruire gratuitamente di un software come javaFX è una perfetta base di partenza.

- Non ci sono problemi di comunicazione tra i software client e server, i dati vengono scambiati in modo semplice e veloce sfruttando il protocollo HTTP.

- La parte dell'applicazione che elabora i dati è trasferita a livello client e fornisce una pronta risposta all'interfaccia utente, mentre la gran parte dei dati e dell'applicazione rimane sul server remoto, con notevole alleggerimento per il computer utente.

Concludendo, abbiamo fuso assieme la forza di LabVIEW nella gestione della strumentazione, con le possibilità offerte da javaFX di realizzare applicazioni web interattive con alti contenuti grafici. Aggiungendo a ciò la possibilità di comunicare utilizzando il protocollo HTTP, i risultati così ottenuti si sono dimostarti pienamente soddisfacenti.

BIBLIOGRAFIA

[1] Giada Giorgi. *Creazione di web server in LabVIEW, creazione di client in AJAX*. Maggio 2010.

[2] Giada Giorgi. *Servizi web in LabVIEW*. Maggio 2010.

[3] M. Bertocco. *Introduzione a LabVIEW*.

[4] L. Benettazzo, G. Giorgi, c. Narduzzi. *dispense di Misure per l'automazione e la produzione industriale*. Dicembre, 2008.

[5] Simon Morris. *JavaFX in Action*. Manning, 2010.

[6] Gail Anderson, Paul Anderson. *Essential JavaFX*. Prentice Hall, June 2009.

[7] Jim Clarke, Jim Connors, Eric Bruno. *JavaFX Developing Rich Internet Application*. Addison-Wesley, 2009.

[8] Lucas L.Jordan. *JavaFX Special Effects*. Apress, 2009.

[9] James L.Weaver, Weiqi Gao, Ph.D., Stephen Chin and Dean Iverson. *Pro JavaFX Platform*. Apress, 2009.

[10] Cay Horstmann. *Concetti di informatica e fondamenti di Java*. Quarta edizione, Apogeo, 2007.

[11] Stefano Vitturi. *Automazione Industriale*. [Pag. 73, architetture di comunicazione].

[12] Andrea Pinazzi. *RESTful Web service*. Maggio, 2010.

[13] http://xhtml.html.it/guide/leggi/51/guida-html.

[14] http://lesim1.ing.unisannio.it/italiano/remlab_ita.html.

[15] http://javascript.html.it/guide/leggi/95/guida-ajax.

[16] http://java.html.it/articoli/leggi/3192/introduzione-a-javafx.

[17] http://download.oracle.com/docs/cd/E17802_01/javafx/javafx/1.3/docs/api.

[18] http://xml.html.it/guide/leggi/100/guida-web-service.

[19] http://asp.html.it/articoli/leggi/673/cookies-e-sessioni.

[20] http://php.html.it/guide/lezione/2664/mantenere-lo-stato-i-cookie.

[21] http://php.html.it/guide/lezione/2665/mantenere-lo-stato-le-sessioni.

[22] http://java.html.it/articoli/leggi/2189/request-e-response-header-del-protocollo-http.