Università degli Studi di Milano

Facoltà di Scienze Matematiche, Fisiche e Naturali

Dipartimento di Informatica e Comunicazione

Corso di Laurea in Scienze e Tecnologie della Comunicazione Musicale

UNIVERSITÀ DEGLI STUDI DI MILANO

CREAZIONE DI TABLATURE A PARTIRE DA XML IN FORMATO IEEE 1599

Relatore: Prof. Luca Andrea Ludovico

Correlatore: Adriano Baratè

Elaborato finale di:

Giacomo Licari

matr. 739173

Anno Accademico 2010/2011

Indice

INTRODUZIONE

Il presente elaborato, nasce dalla necessità di rendere semplice la traduzione delle partiture in tablature per chitarra, senza l'obbligo di dover installare dei software specifici sul proprio computer.
Nel caso più specifico, l'elaborato si pone l'obiettivo di risolvere una partitura per chitarra, esportata nel formato XML IEEE 1599, con la traduzione della stessa in tablatura, resa disponibile tramite un'applicazione Web.
Lo studio si è incentrato prevalentemente sull'analisi del formato specifico dell'XML musicale, dettato dallo standard IEEE 1599, sviluppato dall'Università degli Studi di Milano, e dalla realizzazione di un'applicazione Web, con tegnologie ASP.NET MVC e LINQ.
Ritenendo che il mondo del Web sia in continuo sviluppo, la disponibilità di un'applicazione Web a portata di click rende semplici quelle operazioni editoriali, anche per non addetti ai lavori, che necessiterebbero di software per il raggiungimento degli stessi obiettivi.
Naturalmente, il presente elaborato si presenta a livello accademico, volto comunque a dimostrarsi come un valido punto di partenza che fa ben sperare nel proseguo dell'informatica musicale.

Giacomo Licari

CAPITOLO 1

LINGUAGGI DI MARK-UP ED XML

1.1 Introduzione

L'XML (eXstensible Markup Language) è un meta-linguaggio di markup, approvato dal W3C, che consente di definire a sua volta altri linguaggi di markup.

Storicamente la parola mark-up (marcatura) è stata utilizzata per descrivere commenti o altre indicazioni all'interno di un testo, marcature, che servivano ad istruire un compositore su come distribuire graficamente un particolare passaggio.

Con l'automatizzazione dei processi di formattazione e di stampa dei testi, il termine è stato esteso a tutti i vari simboli relativi alla formattazione, la stampa e l'elaborazione del testo elettronico.

Ogni qualsiasi documento può essere considerato come costituito da **contenuto** e **sistema di contrassegno**. Per contenuto si intendono i caratteri di un testo, le immagini, ovvero qualsiasi cosa veicoli il messaggio del documento, mentre, il sistema di contrassegno, marcatura, ha come obiettivo quello di comunicare informazioni aggiuntive. Tali informazioni arricchiscono il mero contenuto del documento e possono suddividersi nelle categorie di struttura e formattazione.

La struttura definisce una ripartizione logica dei contenuti del documento (ad esempio, capitoli, paragrafi,...), mentre la formattazione ne gestisce l'aspetto (ad esempio, tipo di font, corpo del carattere,...).

Per linguaggio di marcatura (mark-up language) si intende una codifica che utilizza un insieme di marcatori (mark-up) convenzionali, ovvero aderenti a delle chiare regole, definite e comunemente accettate.

Un mark-up language deve specificare:

- il significato di ogni mark-up
- come si distinguono i mark-up dal testo
- quali mark-up sono consentiti
- quali mark-up sono richiesti

L'SGML (Standard Generalized Markup Language) è uno standard internazionale per la descrizione di testi elettronici di tipo mark-up, un meta-linguaggio per la descrizione formale di linguaggi che fornisce gli elementi necessari a soddisfare i primi tre punti.

In SGML un documento viene visto come una struttura gerarchica di elementi annidati.
XML è un sottoinsieme di SGML e ne rappresenta una semplificazione in quanto:

- trascura molte opzioni sintattiche e varianti
- trascura alcune caratteristiche del DTD (Document Type Definition)
- trascura alcune caratteristiche problematiche

I principali vantaggi di XML sono:

- L'intercambiabilità in rete, ovvero la possibilità di comunicare e condividere agevolmente l'informazione codificata, in ambienti distribuiti e su Internet in particolare
- L'indipendenza dalla piattaforma, una caratteristica collegata direttamente al punto precedente
- La struttura gerarchica, con la particolare disposizione a rappresentare l'informazione strutturata secondo schemi gerarchici
- L'intelligibilità, che facilita la lettura dell'informazione codificata
- L'estensibilità, cioè la predisposizione di XML ad aggiunte, integrazioni e modifiche, da operare nel momento in cui il linguaggio si dimostrasse essere carente o inadeguato ai propri scopi
- La disponibilità di tool per l'implementazione del formato e per la validazione dei file prodotti

Ognuno di questi punti appena descritti trova applicazione nel campo musicale, in cui l'informazione è fortemente strutturata e gerarchizzata.

1.2 Gli Elementi

Gli elementi rappresentano le basi su cui si costituisce XML e consentono di attribuire un preciso significato ad una parte di documento.
Ciascun tipo di elemento è rappresentato da un contrassegno, detto anche etichetta (tag).
La sintassi dei tag è la seguente:

```
<element_name>...</element_name>
```

Element_name rappresenta una stringa alfanumerica di lunghezza e contenuti arbitrari.
Notiamo che ogni singolo elemento viene delimitato da una coppia di tag, detti rispettivamente tag di apertura (start tag) e tag di chiusura (end tag). Il tag di chiusura viene identificato dal carattere “/” anteposto all'identificativo nel tag di chiusura.

All'interno di tale coppia di tag si trova il contenuto vero e proprio del tag, che si può considerare un'istanza dell'elemento identificato, nel nostro caso element_name.
Ricordiamo che in XML è possibile definire un'infinità di tag, a seconda delle proprie necessità, l'importante è seguire alcune regole che verranno discusse successivamente.

Ecco un semplice esempio di codice inerente il tag “nota”:

```
   Esempio 1.1
<nota>DO</nota>
<nota>RE</nota>
<nota>MI</nota>
<nota>FA</nota>
<nota>SOL</nota>
<nota>LA</nota>
<nota>SI</nota>
```

Notiamo come sia semplice definire degli elementi ed attribuirgli un contenuto, nel nostro caso il nome della nota.

1.3 I sotto-elementi

Nell'esempio 1.1 abbiamo visto alcune istanze dell'elemento <nota>, sette per la precisione.
Ma fermarci alla sola definizione degli elementi potrebbe essere riduttivo per un linguaggio che permette di lavorare e definire strutture dati molto complesse.
Per tale motivo, i linguaggi di mark-up consentono l'annidamento di elementi a qualsiasi livello. In altre parole, ogni elemento potenzialmente potrebbe contenere al suo interno degli ulteriori elementi, chiamati **sotto-elementi (sub-elements).**

Vediamo subito un esempio.

```
        Esempio 1.2
<accordo>
      <nota>DO</nota>
      <nota>MI</nota>
      <nota>SOL</nota>
      <durata>4/4</durata>
</accordo>
```

Nell'esempio 1.2 vediamo come sia possibile annidare all'interno del tag “accordo” più sotto-elementi, anche diversi tra loro.
Vengono in questo caso definite le note che formano l'accordo e la durata di tale accordo.

Sarebbe stato possibile definire il codice nel seguente modo, perdendo però la struttura gerarchica che ne facilita la lettura e l'estrazione dei dati:

```
          Esempio 1.3
<accordo>DO, RE, MI, 4/4</accordo>
```

Ma questo è soltanto un semplice esempio, volto a far capire come sia possibile manipolare dei file XML a seconda delle proprie necessità.

1.4 Root Tag ed Empty Tag

Tra i diversi elementi visti finora ne esiste uno che ricopre una particolare importanza.
Stiamo parlando dell'elemento document, detto anche **etichetta radice (root tag)**.
Esso è l'elemento più esterno e di conseguenza è quello che racchiude tutti gli altri elementi del documento XML, pertanto deve esistere obbligatoriamente.
Esistono anche una serie di elementi che non presentano l'end tag, essi vengono chiamati empty-tag e non presentano alcun contenuto.
Hanno una forma del tipo <element_name … />.
Il carattere "/" precede la chiusura dello start-tag, mentre i puntini di sospensione in realtà rappresentano un segnaposto per qualcos'altro. Quel qualcos'altro altro non sono che gli attributi che verranno discussi in seguito.
Vediamo un esempio per capire bene l'utilità degli empty tag.

```
         Esempio 1.4
<accordo>
      <nota>
            <nome>DO</nome>
            <alterazione>
                  <diesis />
            </alterazione>
      </nota>
      <nota>MI</nota>
      <nota>SOL</nota>
      <durata>4/4</durata>
</accordo>
```

Nell'esempio 1.4 è stato ripreso il codice illustrato nell'esempio 1.2, ma è stato aggiunto all'interno del tag <nota> l'elemento <alterazione> con al suo interno il sotto-elemento <diesis />.
Innanzitutto vediamo che il tag <accordo> rappresenta il root tag, in quanto contiene al suo interno tutti gli elementi del documento XML.

L'utilizzo dell'empty tag <diesis /> all'interno di <alterazione> ci fornisce delle importanti informazioni sullo stato della nota di nome DO.

1.5 Gli Attributi

Gli attributi vengono utilizzati per aggiungere delle informazioni ad un elemento.
Essi sono sempre associati allo start tag:

```
                        Esempio 1.5
<element_name attribute_1 = "value_1" ... attribute_N = "value_N">
…
</element_name>
```

Un elemento può contenere al suo interno un numero arbitrario di attributi distinti, posti sempre all'interno dello start tag.
Ecco un esempio in cui vengono utilizzati degli attributi per arricchire le informazioni su una nota facente parte di un accordo.

```
            Esempio 1.6
<accordo>
      <nota ottava = "4">DO</nota>
      <nota ottava = "5">MI</nota>
</accordo>
```

In questo esempio viene aggiunto l'attributo "ottava" all'elemento "nota", con l'arricchimento delle informazioni inerenti alla nota di nome DO, da suonare in ottava 4, e la nota di nome MI, da suonare sull'ottava 5.

1.6 Elementi vs Attributi

Come è stato possibile notare, sia gli elementi che gli attributi possono veicolare una certa informazione, ma come è possibile stabilire quando utilizzare un elemento od un attributo per rappresentarla?
Esistono due semplici regole da seguire nella composizione di un documento XML.
Un elemento viene utilizzato quando innanzitutto è fondamentale per il significato del documento ed è debolmente tipato.
Viene utilizzato invece un attributo quando esso non è importante per il significato del documento ed è fortemente tipato.

1.6 Il DTD

Il DTD (Document Type Definition) fornisce un significato standard al fine di descrivere e definire le componenti ammesse nella costruzione di un documento XML, ma anche per tutti i documenti derivati dall'SGML.

Il DTD definisce quali siano i (sub-)elementi che può contenere un elemento, se un elemento può contenere del testo o meno, quali attributi deve contenere e la tipizzazione e defaultizzazione degli attributi, quindi il DTD definisce quali componenti siano lecite all'interno del documento.

Un DTD si compone di due parti:

- Element Type Definition
- Attribute List Declaration

L'Element Type Definition specifica quale debba essere la struttura del documento, i contenuti e gli attributi ammissibili.

```
Esempio 1.7
<!ELEMENT nome-elemento ( modello di contenuto )>
<!ELEMENT A ( B, C ) >
<!ELEMENT A ( #PCDATA ) >

Esempio 1.8
<!ELEMENT nota ( pitch, durata, commenti ) >
<!ELEMENT pitch ( #PCDATA ) >
<!ELEMENT durata ( #PCDATA ) >
<!ELEMENT commenti ( #PCDATA ) >

(#PCDATA sta per Parsed Character Data che indica del testo generico)
```

Il codice XML dell'esempio 1.8 definisce una struttura così composta:

```
<nota>
      <pitch>4</pitch>
      <durata>4/4</durata>
      <commenti>Commento d'esempio</commenti>
</nota>
```

L'Attribute List Declaration concerne la lista e i tip degli attributi permessi per ogni elemento, ovvero tutti quegli attributi che possono essere inseriti all'interno di un determinato elemento, quali elementi possono contenere degli attributi e i valori di default che assume un attributo.

Ogni attributo è specificato da Name, Type ed i valori di default.

- Name definisce un set di attributi relativi ad un dato elemento
- Type stabilisce vincoli sul tipo degli attributi
- Default Value fornisce i valori di default degli attributi

Esistono tre gruppi di tipi d'attributo utilizzabili:

- String types (CDATA, permette l'utilizzo di un qualsiasi testo)
- Tokenized types (ID, IDREF, un insieme fisso di tipi di parole-chiave con significati particolari)
- Enumerated types (specifica il valore di un attributo preso da un elenco specifico di nomi)

```
Esempio 1.9
<! ELEMENT A (#PCDATA) >
 <!ATTLIST A a CDATA #IMPLIED >
 <!ATTLIST A a CDATA #IMPLIED b CDATA #REQUIRED >
 <!ATTLIST A a CDATA #IMPLIED “aaa” >
 <!ATTLIST A a CDATA #REQUIRED “aaa” >
 <!ATTLIST A a CDATA #FIXED “aaa” >
 <!ATTLIST A a (aaa|bbb) #IMPLIED “aaa” >
 <!ATTLIST A id ID #REQUIRED >
 <!ATTLIST A ref IDREF #IMPLIED >
```

Dopo aver distinto le componenti del DTD è necessario parlare di forma e validità del documento, ovvero di Well-Formed e Validity.

Un documento è detto Well-Formed se segue le regole grammaticali dettate del W3C, è detto Valid se è conforme ad un DTD che ne descrive e specifica la struttura.

Il W3C, World Wide Web Consortium è una comunità internazionale in cui le Organizzazioni Membro lavorano insieme per definire e sviluppare uno standard Web, con lo scopo di portarlo al suo massimo potenziale.

CAPITOLO 2

LO STANDARD IEEE 1599

2.1 Introduzione

L'IEEE 1599 è un nuovo formato basato sull'XML il cui scopo è quello di descrivere una serie eterogenea di contenuti musicali. All'interno di un unico file, tutti i simboli musicali, le tracce audio, metadati di catalogo, testi ed elementi grafici inerenti una singola parte musicale sono collegati e sincronizzati reciprocamente all'interno dello stesso framework.

L'IEEE 1599 permette quindi di sincronizzare elementi di natura differente, organizzati in una struttura multi-layer che supporta diversi formati di codifica ed un numero variabile di elementi per layer.

Una prima fase del progetto è iniziata nel 2000, in cui furono fatte valutazioni sulle caratteristiche dei formati per la descrizione della musica esistenti, in particolar modo su SMDL, NIFF ed XML. Seguirono nel 2001 una serie di contatti con altri gruppi di ricercatori in ambito accademico e commerciale. Nel 2002 viene rilasciata una versione prototipo del formato, denominato in origine Musical Application using XML, o semplicemente MAX. Il formato venne discusso a MAX 2002, la prima conferenza internazionale su applicazioni che usano l'XML, organizzata da IEEE CS TC.

Il processo finale di valutazione di IEEE, denominato ballottaggio, si concluse nel Luglio del 2008 con la standardizzazione internazionale di IEEE 1599.

2.2 XML e musica

Come è stato accennato precedentemente, l'IEEE 1599 si basa sull'XML, e ciò è stato scelto per diversi motivi. Innanzitutto, l'XML è un meta-linguaggio formale adatto a rappresentazioni dichiarative, in grado di descrivere entità, quali sono gli oggetti musicali, così come sono, senza l'aggiunta ed il sovraccarico di informazioni inutili.

La rappresentazione avviene in maniera modulare e gerarchica, tuttavia, consente un'interazione tra i vari moduli, interazioni che avvengono in maniera formale ed esplicita.

La modularità dell'XML offre diversi vantaggi per una descrizione ben organizzata e comprensibile. Ogni parte costituisce un'entità separata della descrizione complessiva, pur mantenendo la propria identità. Tuttavia, l'interdipendenza è consentita e resa esplicita da parte del formato basato su XML.

I linguaggi derivati dall'XML sono estensibili, in quanto supportano estensioni ed elementi esterni. Questa prospettiva è importante per ulteriori sviluppi del linguaggio musicale, inoltre, l'XML essendo facile da leggere, da decodificare e modificare, rimane aperto alle modifiche da parte degli utenti.

Nonostante l'XML sia più facile da usare rispetto ai formati binari, ciò non significa che sia possibile gestirlo direttamente, mentre può essere scritto e letto con computer-based systems.

Ad esempio, il Music XML non può essere visualizzato nella sua forma di testo, così da poter essere visualizzato da un normale utente, ma serve uno specifico software per decodificare il formato e rappresentarlo secondo una sequenza di simboli musicali, interpretabili dall'utente.

Concludendo, un liguaggio basato su XML consente leggibilità, estensibilità e durata, per tale motivo è attuo a rappresentare ed organizzare le informazioni musicali in maniera globale.

2.3 Una struttura Multi-Layer

Se si vuole descrivere in maniera completa l'informazione musicale è di fondamentale importanza supportare materiali eterogenei. Grazie appunto alla capacità intrinseca di XML di fornire rigide strutture d'informazione, le rappresentazioni dell'informazione musicale possono essere organizzate in maniera efficace ed efficiente.

IEEE 1599 impiega una struttura Multi-Layer, costituita da sei diversi strati (layers), ognuno dei quali fornisce informazioni su un determinato formato file ed un grado di astrazione diverso per l'informazione musicale. Essi sono i seguenti, mostrati anche in figura 2.1:

- General, metadati relativi alla musica, al brano musicale nella sua interezza, ad esempio catalogo di informazioni relative al pezzo
- Logic, contiene tre sottolivelli, Spine, Los e Layout. Lo Spine contiene la funzione di mappatura spazio-temporale, il Los descrive gli oggetti in partitura ed il Layout si occupa di rappresentare il layout del brano
- Structural, identifica gli oggetti musicali e le loro relazioni reciproche
- Notational, contiene la rappresentazione grafica dello score
- Performance, descrizione computer-based ed esecuzione di musica
- Audio, descrive le proprietà del materiale musicale audio, è anche il livello più basso della struttura di IEEE 1599

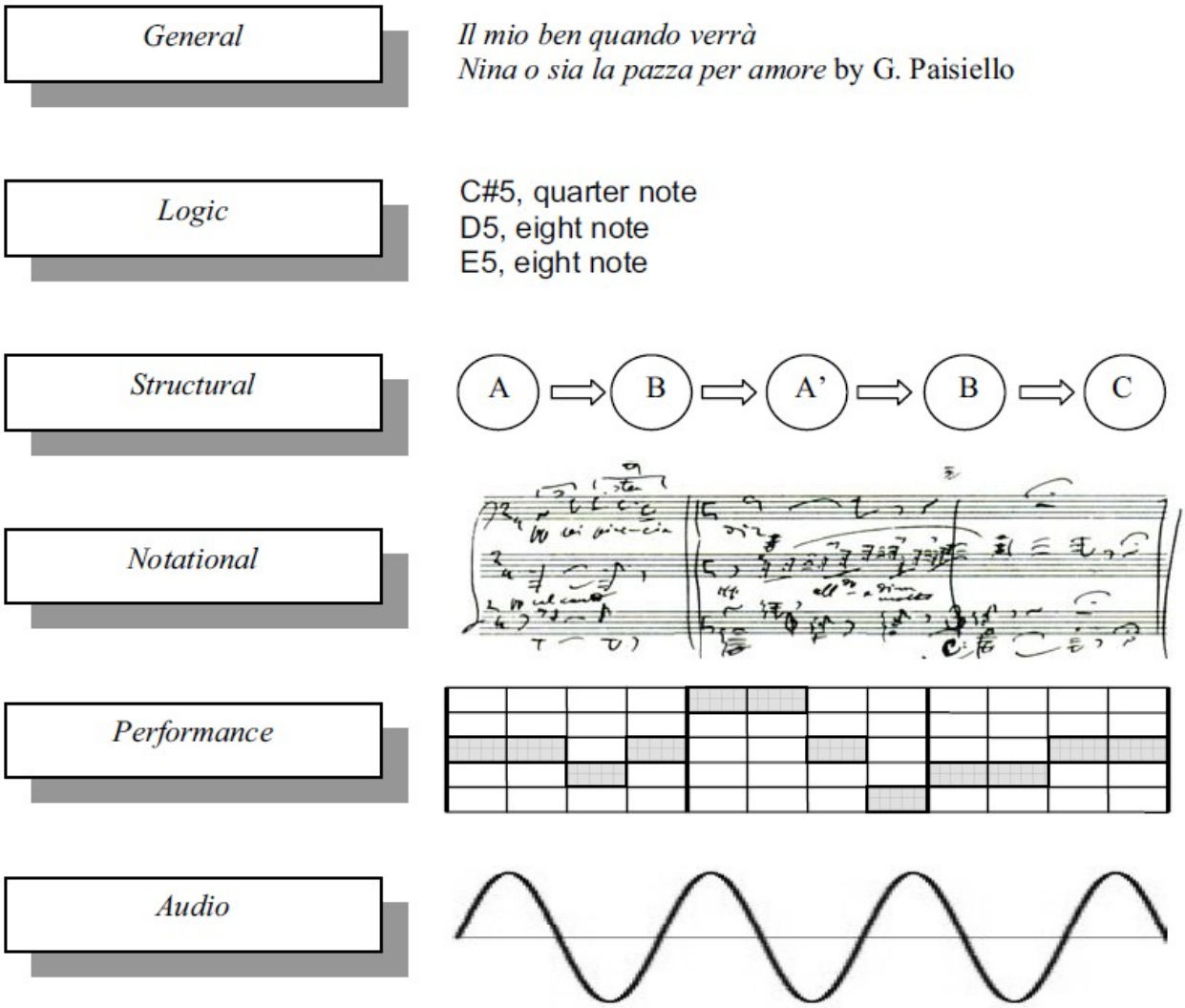


Figura 2.1 – Rappresentazione della struttura Multi-Layer di IEEE 1599

Di seguito, è rappresentato un generico documento in IEEE 1599:

```
Esempio 2.1
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ieee1599 SYSTEM
"http://www.mx.dico.unimi.it/ieee1599.dtd">
<ieee1599>
<general>...</general>
<logic>...</logic>
<structural>...</structural>
<notational>...</notational>
<performance>...</performance>
<audio>...</audio>
</ieee1599>
```

C'è da sottolineare che per un dato brano musicale non è necessario che siano presenti tutti i layer, in ogni modo maggiore è il numero di layer presenti e più ricca sarà l'informazione musicale

presentata. A ciò si aggiunge la possibilità di non utilizzare un singolo file digitale per ogni layer, ma addirittura diversi, contribuendo così all'arricchimento dell'informazione.
Sono due gli approcci di IEEE 1599 alla rappresentazione dell'informazione musicale:

- mantenere le intrinseche descrizioni musicali all'interno del file IEEE 1599
- mantenere gli oggetti multimediali al di fuori del documento XML, nel loro formato d'origine

La figura 2.2 mostra le relazioni esistenti tra il gruppo costituito dai layer General, Structural, Logic e quello che comprende il Notational (ad esempio Gif, Jpeg, Tiff per lo score), l'Audio (ad esempio AAC, MP3, WAV) ed il Performance (ad esempio Csound, MIDI, MPEG).

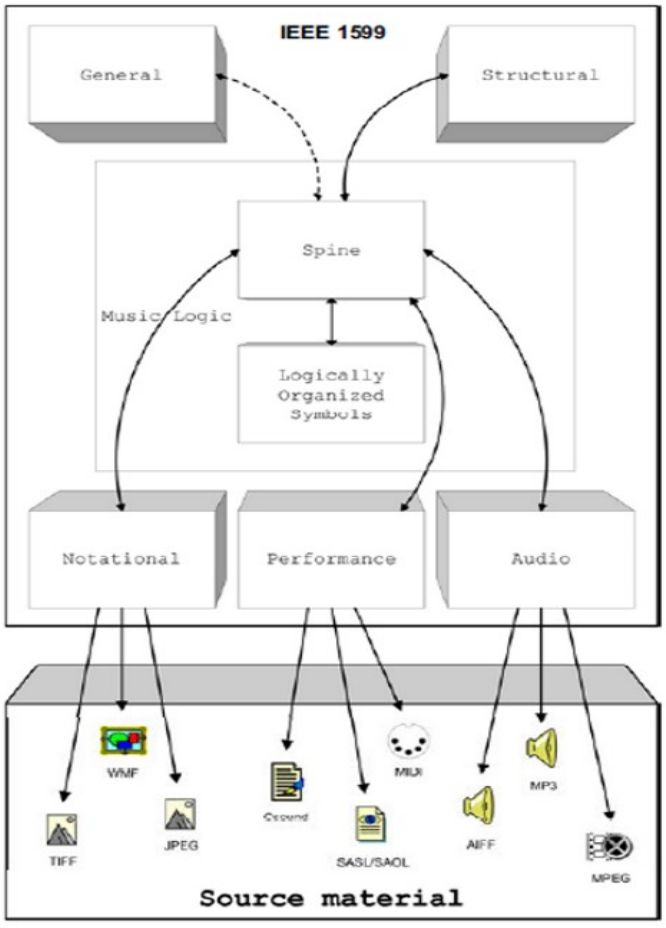


Figura 2.2 – Contenuti codificati all'interno del documento IEEE 1599 ed oggetti multimediali esterni

2.4 Lo Spine

Lo Spine rappresenta una sorta di collante in IEEE 1599, la struttura logica che implementa l'integrazione dei layer notational, performance e audio col layer Logic. Il suo obiettivo è costruire una struttura astratta a cui fanno riferimento tutti gli altri strati che descrivono le proprietà del materiale originario.

Le varie descrizioni del brano musicale stesso non sono semplicemente collegate fra loro, ma è previsto un ulteriore livello di dettaglio:

ogni qualvolta sia possibile, l'informazione dei media è collegata ad un singolo evento musicale.

Un evento musicale può essere definito come il verificarsi, all'interno della partitura, di qualcosa di importante per l'autore della codifica. Ad esempio, le note e le pause di uno score possono essere interpretate come eventi musicali.

Da un punto di vista più generale, in uno score tutti i simboli potrebbero essere eventi musicali, da chiavi a segni di articolazione, nel Jazz, ad esempio, la lista degli eventi potrebbe corrispondere con la griglia armonica.

Lo spine quindi costituisce un elenco di eventi musicali che vengono ordinati e etichettati in modo da consentire le referenze da altri livelli. Comunque, lo score non corrisponde necessariamente alla lista degli eventi referenziati da altri layer.

E' da notare quindi quale sia l'importanza dello Spine che svolge sia la funzione di collante tra i vari layer e che fornisce un livello di astrazione:

è l'autore che può decidere gli elementi da inserire sotto la definizione di evento musicale, a seconda delle esigenze.

Di conseguenza, ciò che è elencato nello Spine deve avere una controparte in qualche strato, altrimenti l'evento non sarebbe definito e la sua presenza nella lista e nel file XML sarebbe completamente inutile. Ad esempio, in un pezzo costituito da N eventi musicali, lo Spine dovrebbe listare N voci senza definirle da nessun punto di vista, come mostrato nell'esempio 2.2.

```
Esempio 2.2
<ieee1599>
<logic>
<spine>
<event id="e1" timing="0" hpos="0"/>
<event id="e2" timing="2" hpos="2"/>
<event id="e3" timing="2" hpos="2"/>
<event id="e4" timing="1" hpos="1"/>
<event id="e5" timing="1" hpos="1"/>
...
</spine>
<los>...</los>
</logic>
</ieee1599>
```

Nell'esempio 2.2 viene presentato un estratto di Spine. Gli eventi listati possono essere definiti semanticamente all'interno del sotto-elemento LOS (Logical Organized Symbols) e referenziati con tutti gli altri layer.
Gli eventi musicali non vengono listati soltanto nello Spine, ma vengono anche marcati con un identificatore univoco. Questi sono indicati da tutte le istanze delle rappresentazioni degli eventi corrispondenti in altri layer.
Ogni elemento dello Spine può essere descritto:

- da 1 a N layer, ad esempio in Logic, Notational e Audio, come i simboli musicali, che hanno una definizione logica, una rappresentazione grafica ed un file audio
- da 1 a N istanze all'interno dello stesso layer, ad esempio in tre differenti clip mappati all'interno del layer Audio
- da 1 a N occorrenze all'interno della stessa istanza, ad esempio, le note in un ritornello che viene eseguito 4 volte, quindi gli eventi dello Spine vengono mappati 4 nel layer Audio, in tempi diversi

Grazie allo Spine, l'IEEE 1599 non è soltanto un semplice contenitore di elementi eterogenei relativi ad un unico pezzo musicale. Si dimostra invece, che tali media possono presentare una serie di riferimenti ad una struttura comune.

CAPITOLO 3

LE TABLATURE

3.1 Introduzione

L'intavolatura o tabulatura, dal latino tabula, o ancora tablatura, è un sistema di notazione secondo cui viene rappresentata graficamente il posizionamento delle dita di una mano per ottenere un suono su uno stumento musicale.

Le tablature sono fortemente diffuse in ambito chitarristico, e oltre a rappresentare un mezzo che rende possibile l'esecuzione musicale da chi non è a conoscenza della notazione musicale tradizionale, rappresenta anche un approccio diverso per fissare un testo musicale.

3.2 Storia ed origini

Le origini dell'intavoltaura risalgono al 1300 circa, i primi manoscritti conosciuti sono in intavolatura tedesca per organo. Esistono esempi a stampa dal 1500 circa, come testimonia l'"Intavolatura de Lauto" di Francesco Spinacino (Venezia 1507) stampata a Venezia da Ottaviano Petrucci mediante la tecnica a caratteri mobili metallici.

Esistono moltissime testimonianze di composizioni per vari strumenti a tastiera e a corda in intavolatura, non soltanto per chitarra, liuto o strumenti simili, ma anche per il violino. Ne "Il scolaro, per imparare a suonare di violino et altri strumenti" di Gasparo Zannetti (Milano 1645), edito in edizione anastatica, si trova un esempio di intavolatura per violino con tecnica di stampa a caratteri mobili. La corda più acuta è posizionata in basso, e sopra il rigo troviamo i valori di durata.

In "Varii scherzi di sonate per la chitarra Spagnola" di Franceso Corbetta (Bruxelles 1648) troviamo invece un esempio di intavolatura per chitarra a 5 corde, incisa con la tecnica della calcografia.

Anche in questo caso i valori relativi alle note sono posizionati in alto sopra al sistema in corrispondenza del posizionamento del primo cambio di valore.

Le intavolature erano anche utilizzate per gli strumenti a tastiera, organo e clavicembalo, soprattutto per quanto riguarda l'intavoltura tedesca, ma anche per fisarmonica, armonica, salteri e molti altri strumenti.

Anticamente, il valore delle note veniva posizionato sopra le note stesse, in corrispondenza di ogni cambio di valore e generalmente venivano impiegati i seguenti simboli:

Figura 3.1

Questo genere di intavolature (tablature) variavano sia nella grafia che nei sistemi a seconda dell'area geografica. Vengono ritenute le più importanti quelle Italiana e Spagnola, oltre che a quelle Francesi e Tedesche.

Le intavolature italiane e spagnole utilizzavano un numero di linee pari a quella delle corde presenti sullo strumento e presentavano dei numeri per indicare quali tasti premere.

La differenza consisteva nell'ordine di rappresentazione delle corde:

nell'intavolatura italiana, la corda più acuta veniva indicata sulla linea inferiore, mentre in quella spagnola la corda più acuta veniva rappresentata sulla linea superiore.

I tasti erano numerati in ordine di semitono, partendo dalla corda vuota, detta anche capotasto, che era segnato con lo “0” (zero).

Ecco un esempio tratto da A. Rotta, “Intabolatura de Lauto, libro primo” (Venezia 1546) e la sua trascrizione in notazione moderna:

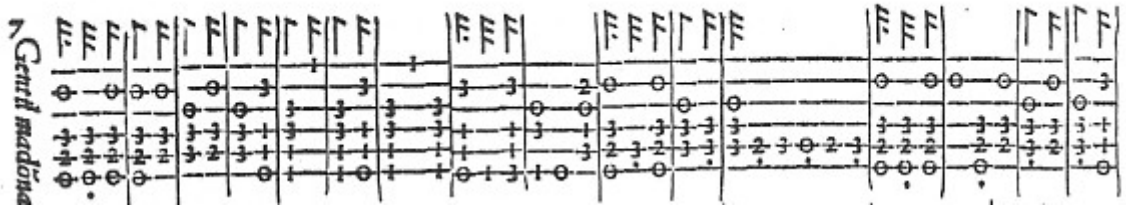


Figura 3.2

A differenza delle intavolature italiane e spagnole, quella francese sostituiva i numeri con le lettere dell'alfabeto. Per cui al capotasto si faceva corrispondere la lettera A, al primo tasto la lettera B e così via, mentre, per quanto riguarda la disposizione delle corde, come per l'intavolatura spagnola, la corda più acuta corrispondeva alla linea superiore.

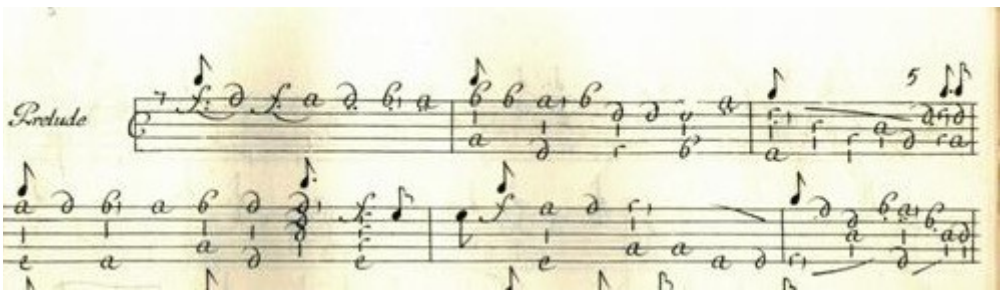


Figura 3.3

Le intavoltature tedesche differivano completamente dalle altre, esse non presentavano delle linee per rappresentare le corde ed inoltre utilizzavano sia numeri che lettere dell'alfabeto.

I numeri indicavano le corde vuote, mentre le 23 lettere dell'alfabeto gotico indicavano i tasti, con un sistema che associava ad ogni tasto di ogni choro un unico simbolo. Per finire tutta la serie di suoni disponibili si riprendevano le prime lettere sormontate da un trattino orrizzontale. In un secondo tempo fu introdotto negli strumenti un sesto choro, e per mantenere il sistema già adottato nell'intavolatura furono escogitate varie soluzioni, tra le quali quella più utilizzata adottava il numero 1 barrato per indicare la corda vuota, e le lettere maiuscole per indicare i tasti successivi.

3.3 Le tablature oggi

Oggigiorno le intavolature vengono ancora utilizzate come sistema alternativo alla notazione tradizionale musicale, soprattutto per la musica popolare, rock e per la chitarra elettrica e il basso. Nelle moderne edizioni al posto della chiave viene posizionata l'abbreviazione TAB, con lo scopo di identificare l'utilizzo del rigo musicale (non l'altezza delle note). Ogni nota è individuata dal posizionamento del numero del tasto sulla corda appropriata, e ciò rispecchia l'antica intavolatura spagnola.

Questo tipo di intavolatura moderna, a differenza di quelle antiche, non permette l'identificazione dei valori ritmici delle note. Pertanto, generalmente viene utilizzato un pentagramma con la notazione tradizionale, posto sopra l'intavolatura, e i due pentagrammi sono uniti fra loro mediante le systemic barline.

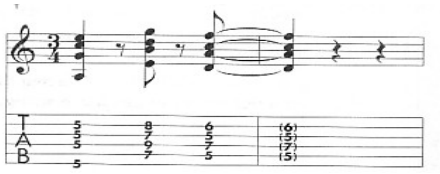


Figura 3.4

CAPITOLO 4

ASP.NET MVC

4.1 Introduzione

ASP.NET MVC è un framework per la realizzazione di pagine Web che utilizza la piattaforma Microsoft ASP.NET MVC (acronimo di Model View Controller).

In termini astratti, una pagina Web può essere vista come un contenitore duale, dove un'interfaccia pubblica è supportata da diverse tecnologie su una varietà di piattaforme hardware/software. Pubblicamente, una pagina Web produce una miscela di markup standard costituito da HTML, Cascading Style Sheet (CSS) e JavaScript che i Web Browser riescono a visualizzare. Da un punto di vista interno, una pagina Web può impiegare diverse tecnologie, framework, linguaggi e pattern per trasformare una richiesta Web in una buona miscela di markup.

Nel 2001 Microsoft introdusse nel mondo del Web Development la piattaforma ASP.NET che aprì le porte dello sviluppo Web ad un enorme numero di professionisti ed inoltre contribuì al cambiamento del modello di sviluppo delle applicazioni Web.

Nella sua opera rivoluzionaria, ASP.NET però seguiva il progresso raggiunto da due precedenti tecnologie: Active Server Pages (ASP) classico e Java Server Pages (JSP).

4.2 Il profondo impatto di ASP.NET

L'ASP classico aveva due principali caratteristiche. In primis, rendeva la generazione dinamica di HTML realmente facile a molti sviluppatori. Secondo, è stato uno dei primi ambienti di programmazione ad ospitare la logica delle applicazioni Web nel Web server con un ulteriore notevole guadagno di prestazioni.

Le pagine ASP, basate su codice script ed interpretate da un motore runtime, vennero elevate al rango del codice compilato con l'avvento della piattaforma .NET. Oggigiorno l'ASP classico è stato totalmente soppiantato da ASP.NET, ma sopravvive ancora in alcuni pochi vecchi siti Web.

Le pagine ASP.NET si basano su codice compilato scritto utilizzando veri e propri linguaggi di programmazione come Microsoft C# e Visual Basic. Se realizzare codice con ASP era realmente facile, oggi lo è di più ed anche in maniera più uniforme con ASP.NET.

4.3 Adattamento del modello RAD al Web

In Microsoft, prima che venisse sviluppato ASP.NET, la best practice era il modello RAD, a eventi, di Visual Basic. Ciò rendeva rapido e facile prototipizzare un'applicazione che necessitava di un'interfaccia utente. Si partiva sistemando alcuni pulsanti su un form, e con un doppio click su di essi veniva aggiunto uno stub di codice, per poi modificare quel codice con qualche comando database.

Il modello RAD Visual Basic fu creato per applicazioni desktop smart-client. La difficoltà per il team di sviluppo ASP.NET fu proprio quella di come fare ad estendere il modello RAD al Web.

Il modello RAD originale si evolse successivamente nel modello Windows Forms, con l'introduzione del Microsoft .NET framework. Con Windows Forms, non importava che connettività esistesse tra i componenti client e server, il server agiva sempre in reazione all'input del client. Il server è in questo contesto consapevole dello stato complessivo dell'applicazione e opera in modo two-tier, connesso. Questo meccanismo è stato facile da implementare in uno scenario smart-client, ma ha richiesto qualche meccanismo extra per farlo operare sul Web.

4.4 L'età della ragione di ASP.NET

ASP.NET oggi viene adottato quasi in ogni progetto Web rivolto alla piattaforma Microsoft, comunque, cinque anni di una tecnologia software costituiscono un enorme lasso di tempo.

Qualsiasi tecnologia software mostra inevitabilmente i primi segni dell'età dopo così tanto tempo, anche in virtù di nuove piattaforme concorrenti, e ASP.NET non fa eccezione.

Microsoft, nel corso degli anni ha raffinato ASP.NET, offrendo una sofisticata piattaforma per lo sviluppo AJAX ed i controlli sono stati adattati per supportare meglio i requisiti CSS e XHTML.

4.5 Uno scorcio su ASP.NET MVC

ASP.NET MVC è una nuova piattaforma per la costruzione di applicazioni ASP.NET. Essa si basa sullo stesso ambiente run-time del classico ASP.NET Web Forms, ma rende lo sviluppo delle applicazioni una esperienza notevolmente differente rispetto al vecchio modello Web Forms.

ASP.NET MVC è stato concepito per concentrarsi sulle azioni eseguibili da un utente nell'esplorare una pagina. Ha un differente motore di visualizzazione e permette un maggiore controllo sul markup generato. ASP.NET MVC non tiene conto del pattern Page Controller e opta per un pattern differente che può essere considerato una variante orientata al Web del classico pattern Model-View-Controller (MVC)

4.5.1 Punti salienti di ASP.NET MVC

ASP.NET MVC introduce un sistema di sviluppo completamente nuovo, in cui il programmatore nel scrivere l'applicazione ragiona in termini di controller e di visualizzazioni. Le proprie decisioni riguardano come esporre i propri controller agli utenti e come passare i dati alla visualizzazione. Ogni richiesta viene risolta invocando un metodo di una classe controller. Non è mai necessario nessun postback per servire una richiesta utente, e nessun view state è mai necessario per mantenere lo stato della pagina. Infine, non esistono controlli server per produrre HTML per il browser.
In verità, se si fa un po' più di attenzione dietro le quinte di ASP.NET MVC, è chiaro comunque che il suo modo di lavorare sia ancora basato sulla gestione delle richieste HTTP (HyperText Tansfer Protocol), eccetto che la stringa URL viene trattata diversamente e che ogni azione risultante viene espressa dagli sviluppatori utilizzando i metodi delle classi controller invece dei postback.

4.5.2 Il pattern sottostante

Il meccanismo di funzionamento di ASP.NET MVC si basa sulla combinazione di due pattern:
il pattern Front Controller e il pattern Model2. Insieme, questi due pattern propongono un modello di programmazione notevolmente differente da ASP.NET Web Forms.
Il pattern Front Controller comporta l'utilizzo di un componente centralizzato che gestisce tutte le richieste in ingresso e le invia a un ulteriore componente successivo della pipeline per servire effettivamente la richiesta. Così tutte le richieste vengono incanalate attraverso un unico componente. In ASP.NET MVC questo componente è l'handler HTTP MVC.
Questa classe comune contiene la logica che analizza l'URL e decide a quale controller spetta servire la richiesta e a quale componente View spetta produrre l'HTML risultante. Il controller è una semplice classe con metodi pubblici e ciascuno di essi esegue un'azione conseguente alle azioni dell'utente.

La Figura 4.1, visionabile alla pagina seguente, illustra l'approccio Front Controller.

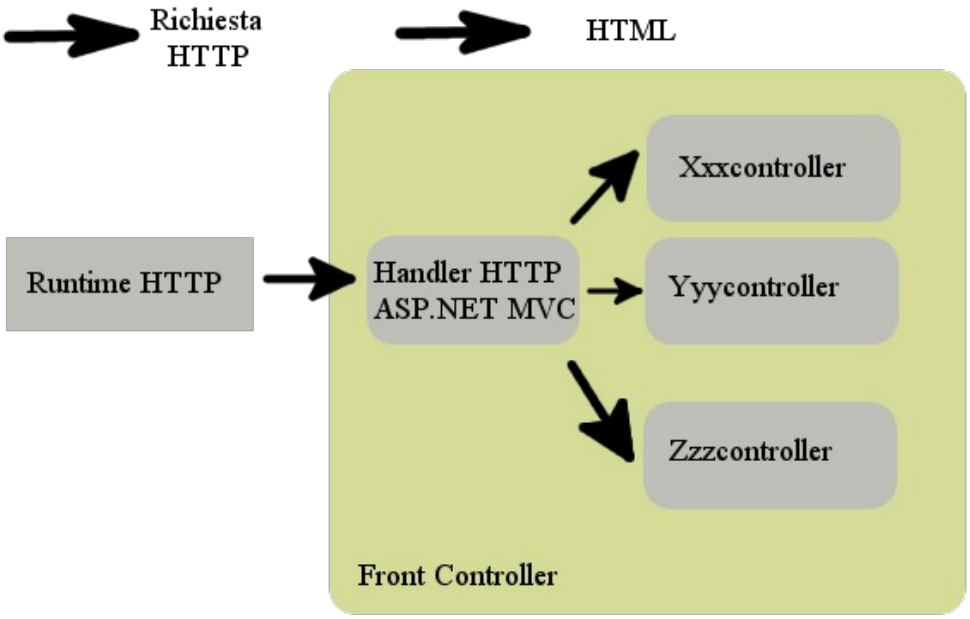


Figura 4.1 – Approccio con Front Controller

In ASP.NET MVC, l'interazione tra il Front Controller e i controller specifici all'azione e le visualizzazioni è regolamentata dal pattern Model2.

Il Front Controller stabilisce il controller da utilizzare e invoca uno dei relativi metodi. Il metodo del controller viene eseguito, riceve alcuni dati e stabilisce la visualizzazione da utilizzare. Infine, la visualizzazione genera il markup per il browser e lo scrive nello stream di output della risposta.

4.6 I controller

L'obiettivo primario del pattern MVC è separare il più nettamente possibile il codice che genera l'interfaccia grafica visualizzata agli utenti dal codice che gestisce le azioni dell'utente.

In una applicazione ASP.NET MVC, ogni richiesta che giunge sul Web server viene intercettata dal modulo di routing e viene inviata a un handler HTTP centralizzato: l'handler HTTP MVC.

Quest'ultimo, a sua volta, esamina il contenuto della richiesta (precisamente, il formato dell'URL) e stabilisce il controller da utilizzare. Ne risulta che in ASP.NET MVC non ci sia nessun ciclo di vita della pagina. L'handler HTTP che si incarica della richiesta è unico e non è specifico per la pagina.

Per quanto riguarda la progettazione delle applicazioni, non si ragiona tanto in termini di pagine da creare e di codice, piuttosto, ci si concentra sulle azioni che un utente potrebbe eseguire sull'interfaccia ad egli visualizzata.

Ricordiamo che il controller è una semplice classe con alcuni metodi pubblici. Ognuno di questi metodi ha solitamente un collegamento uno a uno con una possibile azione dell'utente, come ad esempio la visualizzazione dei dettagli di un oggetto, la modifica della selezione in una lista.
E' da notare quindi come il ruolo dei controller sia centrale nell'architettura MVC.

4.6.1 Azioni e controller

In ASP.NET MVC ciascuna richiesta è in definitiva diretta all'esecuzione di un metodo di una classe controller selezionata. Il metodo del controller viene eseguito, elabora i dati di input, esegue qualche logica applicativa e stabilisce la vista da utilizzare.

Di seguito la struttura tipica di una classe controller con due metodi.

```
Esempio 4.1
public class HomeController : Controller
{
      public ActionResult Index()
      {
            // Esegue qualche logica applicativa
            .
            .
            .
            // Cede il controllo al motore di visualizzazione.
            // Il nome della visualizzazione in questo caso è per
            // default il nome del metodo.
            Return this.View();
      }
      public ActionResult About()
      {
            //Esegue qualche logica applicativa
            .
            .
            .
            // Cede il controllo al motore di visualizzazione.
            // Il nome della visualizzazione viene specificato
            // esplicitamente.
            Return this.View("About");
      }
}
```

Un metodo di un controller è previsto che restituisca un oggetto ActionResult o un oggetto che derivi da ActionResult.

4.7 Le visualizzazioni

In ASP.NET MVC, una visualizzazione è una classe che riceve un template e alcuni dati e poi produce una risposta per il browser. Il controller seleziona la visualizzazione successiva e le chiede di renderizzare la risposta. Il controller non riceverà nulla in risposta dalla visualizzazione. Le responsabilità del controller terminano quando cede il controllo alla visualizzazione. Di conseguenza, la visualizzazione è responsabile della scrittura nello stream di output di qualsiasi contenuto per il browser.

Una visualizzazione non ha una vita autonoma all'interno del framework, ma esiste solo per essere invocata al termine di certe azioni del controller per produrre una risposta.

Il componente View, visto nell'esempio precedente, è la classe che lo sviluppatore scrive per completare il mosaico che ha come risultato che l'ambiente runtime fornisca una risposta HTML al browser. Se si guarda dietro le quinte dell'oggetto view, si nota però un meccanismo piuttosto complesso che ruota attorno all'oggetto view engine (motore di visualizzazione).

Il motore di visualizzazione è il componente che costruisce fisicamente l'output per il browser. Il motore prende uno specifico file template di motore e ne mescola il contenuto con qualsiasi informazione di contesto riceva dal controller.

L'output finale generato da un motore di visualizzazione è previsto che sia soprattutto HTML, ma si potrebbe trattare anche di qualcosa di diverso, come uno speciale tipo di contenuto, stringhe JSON, JavaScript, per questo probabilmente si farà meglio a utilizzare un tipo di action result ad hoc.

Il view engine è un elemento plug-in dell'architettura ASP.NET MVC ed utilizza la familiare sintassi di markup ASPX di ASP.NET Web Forms. Ciò significa che si possono riutilizzare diverse caratteristiche del markup Web Forms, compreso i controlli server, le pagine master, i temi.

4.8 I modelli

In ASP.NET MVC, il modello è semplicemente un'astrazione dei dati che il controller passa alla visualizzazione e lo stesso termine “modello” viene utilizzato per esprimere tre cose distinte:

- La rappresentazione dei dati che vengono postati dal controller
- La rappresentazione dei dati che vengono lavorati nella visualizzazione
- La rappresentazione delle entità specifiche del dominio che operano nel livello business

In uno scenario più semplice, potrebbe presentarsi ed essere impiegato un solo ed unico insieme di classi, quindi un unico modello, in gran parte desunto dal database.
In scenari più sofisticati, dove essenzialmente si hanno più livelli nel server, è importante riconoscere la differenza tra il modello di dominio ed il view-model.
Il modello di dominio è la rappresentazione dei dati che si crea a vantaggio dell'elaborazione business. Il view-model è la rappresentazione dei dati che si crea a vantaggio della visualizzazione, una collezione di classi, ciascuna che rappresenta l'insieme di dati su cui una determinata visualizzazione lavorerà.
La visualizzazione riceve solo gli oggetti view che contengono una rappresentazione dei dati che soddisfa solo le necessità della visualizzazione.

CAPITOLO 5

LINQ

5.1 Introduzione

Dato che Microsoft .NET e i suoi linguaggi per C# e VB sono maturati, è diventato evidente che uno dei maggiori problemi ancora presenti per gli sviluppatori è quello di accedere a dati provenienti da diverse fonti. In particolare, l'accesso al database, la manipolazione degli XML.

Linq è la tecnologia sviluppata da Microsoft per fornire un meccanismo language-level a sostegno dell'interrogazione di tutti i tipi di dati. Questi tipi includono array e collections, database, documenti XML ed altro.

5.2 Linq e le Query

Per la maggior parte, Linq è incentrato sulle query, sia che esse siano un insieme di oggetti, un unico oggetto, un sottoinsieme di campi estratti da un oggetto o un insieme di oggetti. Il risultato di una query in Linq si chiama sequenza, la cui maggiorparte sono di tipo IEnumerable<T>, dove T è il tipo di dati memorizzati nella sequenza.

```
Esempio 5.1
using System;
using System.Linq;
using System.Data.Linq;

Unimi db = new Unimi(@"Data Source=.\SQLEXPRESS;InitialCatalog=Unimi");
var students = from c in db.Studenti
where c.Nome == "Giacomo"
select c;

foreach (var x in students)
{
      Console.WriteLine("{0}", c.Cognome);
}
```

Nell'esempio 5.1 notiamo una semplice query LINQ to SQL in cui interroghiamo un database.

Innanzitutto bisogna aggiungere delle direttive assembly al progetto, affinchè si possano utilizzare le funzionalità di LINQ to SQL:
using System.Linq, using System.Data.Linq.
Viene utilizzata la keyword var per ragioni di semplicità.
Con il cilco foreach estraiamo tutti i valori che risultano soddisfare i criteri di ricerca impostati nella query, in particolar modo andiamo ad individuare tutti i cognomi degli studenti, il cui nome è Giacomo.

5.3 LINQ to XML

Dopo anni di esperienza con Microsoft W3C XML DOM API, diverse settori chiave sono stati individuati e classificati come inconvenienti, pertanto al fine di combattere tali defezioni, sono stati trattati i seguenti punti:

- XML tree construction
- Document centricity
- Namespaces e prefissi
- Estrazione del valore dei Nodi

Ognuno di questi punti è stato sempre un blocco nel lavorare con XML, ma con LINQ to XML questi problemi sono stati risolti.

```
Esempio 5.2
XElement xSpartito =
new XElement("Spartito",
new XElement("Titolo", "Tres notas para decirte te quiero"),
new XElement("Autore", "Vicente Amigo"));

Console.WriteLine(xSpartito.ToString());
```

Con LINQ ecco che la costruzione di un file XML diventa davvero semplice.
Innanzitutto si dichiara l'elemento, a cui viene dato il nome il nome xSpartito, di tipo XElement, ovvero la classe che rappresenta un elemento XML.
Nell'esempio, quando viene dichiarato l'elemento Spartito, al suo interno inseriamo due oggetti Xelement con i loro valori, che vanno a formare gli elementi figli di Spartito.

Ecco il risultato in output:

```
<Spartito>
      <Titotolo>Tres notas para decirte te quiero</Titolo>
      <Autore>Vicente Amigo</Autore>
</Spartito>
```

Altrettanto semplice risulta l'estrazione dei valori dai nodi di un documento XML:

```
Esempio 5.3
XElement nome = new XElement("Nome", "Giacomo");
Console.WriteLine(nome);
Console.WriteLine((string)nome);
```

Il risultato prodotto dall'esempio 5.3 è il seguente:

```
<Nome>Giacomo</Nome>
Giacomo
```

Per estrarre il valore di un nodo si può semplicemente eseguire il cast del nodo secondo il tipo di dato cui il valore può essere convertito.
Nell'esempio precedente abbiamo eseguito il cast string sul nodo, ma possiamo utilizzare tutti i tipi di dati disponibili, come nell'esempio seguente:

```
Esempio 5.4
XElement number = new XElement("Numero", 12);
Console.WriteLine(number);
Console.WriteLine((int)number);

XElement smoker = new XElement("Fumatore", false);

Console.WriteLine(smoker);
Console.WriteLine((bool)smoker);

XElement pigreco = new XElement("Pigreco", 3.1415926535);

Console.WriteLine(pigreco);
Console.WriteLine((double)pi);
```

Dall'esempio 5.4 in output si ottiene:

```
<Numero>12</Numero>
12

<Fumatore>false</Fumatore>
false
```

```
<Pigreco>3.1415926535</Pigreco>
3.1415926535
```

Non solo è semplice creare elementi XML ed seguire query su di essi, ma è possibile effettuare una serie di operazioni come la creazione di attributi grazie alla classe XAttribute.
L'utilizzo di XAttribute è semplice come l'utilizzo di XElement. Definito l'elemento, si nomina al suo interno l'attributo, così come avviene per la creazione degli elementi figli.
Ecco un esempio:

```
Esempio 5.5
XElement xBookParticipant = new XElement("BookParticipant",
new XAttribute("type", "Author"));
Console.WriteLine(xBookParticipant);
```

In output al seguente codice si ottiene la seguente stringa:

```
<BookParticipant type=”Author” />
```

A volte può capitare di non creare l'elemento e l'attributo allo stesso tempo, per questo è possibile istanziare un elemento dapprima per poi aggiungergli un attributo in un secondo momento.

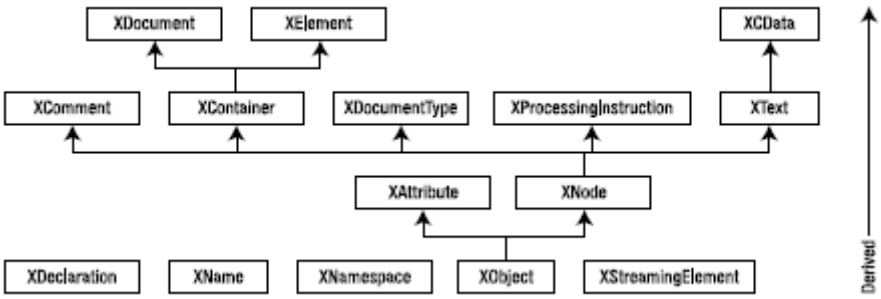
```
Esempio 5.6
XElement xBookParticipant = new XElement("BookParticipant");
XAttribute xAttribute = new XAttribute("type", "Author");
xBookParticipant.Add(xAttribute);

Console.WriteLine(xBookParticipant);
```

Il risultato ottenuto dall'esempio 5.6 è identico a quello ottenuto dall'esempio 5.5:

```
<BookParticipant type=”Author” />
```

Nella figura 5.1 viene mostrato lo schema delle classi utilizzate da LINQ to XML.



5.4 XML output

Con LINQ to XML è possibile salvare un documento usando uno dei tanti metodi XDocument.Save.

Dopo avere creato un documento con Xdocument, si può precedere al salvataggio dello stesso.

Di seguito alcuni esempi:

```
Esempio 5.7
void  Xdocument.Save(string filename);
void Xdocument.Save(XmlWriter writer);
void Xdocument.Save(string filename, SaveOptions options);

Esempio 5.8
XDocument spartito = new XDocument(
new XElement("Spartiti",
      new XElement("Spartito",
      new XElement("Autore", "Giacomo Licari"),
      new Xelement("Titolo", "Il mio brano")));

spartito.Save("spartiti.xml");
```

Nell'esempio 5.8 è stata innanzitutto dichiarata la struttura del fil XML e successivamente è stato salvato il file.

Aprendo il file con un browser il risultato ottenuto è il seguente:

```
Esempio 5.9
<?xml version="1.0" encoding="uft-8"?>
<Spartiti>
      <Spartito>
            <Autore>Giacomo Licari</Autore>
            <Titolo>Il mio brano</Titolo>
      </Spartito>
</Spartiti>
```

Nell'esempio 5.9 viene mantenuta la formattazione gerarchica dell'XML, ma se volessimo eliminare la visualizzazione così come mostrata dall'esempio, avremmo dovuto aggiungere la seguente opzione al metodo Save:

```
spartito.Save("spartiti.xml", SaveOptions.DisableFormatting);
```

Il risultato sarebbe simile al seguente:

```
Esempio 5.10
<?xml version=”1.0” encoding=”uft-8”?><Spartiti><Spartito><Autore>Giacomo
Licari</Autore><Titolo>Il mio brano</Titolo></Spartito></Spartiti>
```

Le stesse funzionalità estendibili dalla classe XDocument sono utilizzabili dalla classe XElement, compreso il metodo Save.

Nel seguente esempio viene utilizzata la classe XElement al posto di XDocument.

```
Esempio 5.11
XElement spartito = new XElement(
new XElement(“Spartiti”,
      new XElement(“Spartito”,
      new XElement(“Autore”, “Giacomo Licari”),
      new Xelement(“Titolo”, “Il mio brano”)));

spartito.Save(“spartiti.xml”);
```

Il risultato in output ottenuto dal file spartiti.xml è identico al file spartiti.xml creato con la classe XDocument.

```
Esempio 5.12
<?xml version=”1.0” encoding=”uft-8”?>
<Spartiti>
      <Spartito>
            <Autore>Giacomo Licari</Autore>
            <Titolo>Il mio brano</Titolo>
      </Spartito>
</Spartiti>
```

5.5 XML input

LINQ to XML non solo prevede la creazione di documenti XML in maniera molto semplice, ma mette a disposizione la possibilità di caricare un documento XML esterno, sul quale si potranno eseguire query e modifiche.

La classe che permette il caricamento di un file XML è la XDocumente.Load e mette a disposizione una varietà di metodi, mostrati nella pagina seguente.

```
Esempio 5.13
static XDocument XDocument.Load(string uri);
static XDocument XDocument.Load(TextReader textReader);
static XDocument XDocument.Load(XmlReader reader);
static XDocument XDocument.Load(string uri, LoadOptions options);
static XDocument XDocument.Load(TextReader textReader, LoadOptions options);
static XDocument XDocument.Load(XmlReader reader, LoadOptions options);
```

Si può notare come questi metodi siano simmetrici ai metodi XDocument.Save. Tuttavia, ci sono un paio di differenze che vale la pena sottolineare. In primo luogo, per i metodi di Save, è necessario richiamare il metodo Save in un oggetto di tipo XDocument o XElement perchè il metodo Save è un metodo di istanza.

Invece il metodo Load è statico, pertanto lo si può richiamare sulla classe XDocument stessa. Inoltre, i metodi Save che accettano una stringa richiedono il nome da assegnare al file da salvare, mentre nei metodi Load le stringhe contengono l'indirizzo URI del file da caricare.

Esistono alcuni parametri addizionali che vanno a completare il metodo Save come esposto nella seguente tabella.

OPTION	DESCRIZIONE
LoadOptions.None	Specifica che non ci sono opzioni da usare
LoadOptions.PreserveWhitespace	Questa opzione preserva gli spazi bianchi nel sorgente XML, come ad esempio le righe vuote
LoadOptions.SetLineinfo	Si utilizza questa opzione in modo da ottenere la linea e la posizione di qualsiasi oggetto che eredita dalla XObject utilizzando l'interfaccia IXmlLineInfo
LoadOptions.SetBaseUri	Questa opzione permette di ottenere l'URI di base di qualsiasi oggetto che eredita da XObject

Di seguito un esempio che dimostra il caricamento di un file XML ed il risultato ottenuto dal comando Console.WriteLine.

```
Esempio 5.14
XDocument xDocument = XDocument.Load("bookparticipants.xml",
LoadOptions.SetBaseUri | LoadOptions.SetLineInfo);
Console.WriteLine(xDocument);

<BookParticipants>
<BookParticipant type="Author" experience="first-time" language="English">
<FirstName>Joe</FirstName>
<LastName>Rattz</LastName>
</BookParticipant>
</BookParticipants>
```

CAPITOLO 6

VEX FLOW, MUSIC ENGRAVING IN JAVASCRIPT

6.1 Cos'è Vex Flow?

Vex Flow è una notazione musicale web-based open source, può essere usato come backend di rendering per vari tipi di tools musicali online, librerie ed applicazioni.

E' scritto completamente in JavaScript, supporta i Canvas di HTML5 e SVG. I Canvas sono elementi di HTML5 che permettono il rendering tramite script di forme 2D e di immagini bitmap.

Gli SVG (Scalable Vector Graphic), invece, sono una famiglia di specifiche basate su XML per la descrizione di vettori grafici a due dimensioni, sia statici che dinamici, ovvero interattivi o animati.

E' importante notare che Vex Flow è un render API di basso livello, la maggiorparte delle applicazioni useranno Vex Tab che è un linguaggio di alto livello per il rendering di tablature per chitarra e di notazione musicale.

6.2 Vex Tab

Vex Tab è un linguaggio che permette di creare, modificare e condividere tablature per chitarra. A differenza dei tab ASCII, progettati per aumentarne la leggibilità, VexTab è stato progettato per migliorarne la scrivibilità.

Nell'immagine seguente viene presentato un esempio di tablatura creata con VexTab.

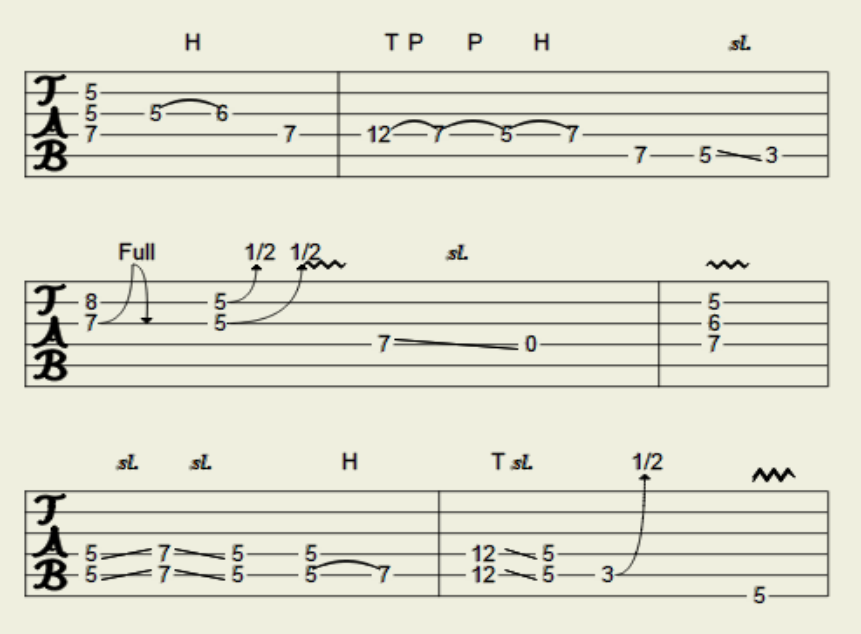


Immagine 6.1 – Tablatura scritta con Vex Tab

La tablatura mostrata nell'immagine 6.1 è stata generata dal seguente semplice codice:

```
Esempio 6.1
tabstave
notes (5/2.5/3.7/4) 5h6/3 7/4 |
notes t12p7p5h7/4 7/5 5s3/5

tabstave
notes (8/2.7b9b7/3) (5b6/2.5b6/3)v 7s0/4 |
notes (5/2.6/3.7/4)v

tabstave
notes (5/4.5/5)s(7/4.7/5)s(5/4.5/5) (5/4.5/5)h(7/5) |
notes t(12/5.12/4)s(5/5.5/4) 3b4/5 5V/6
```

La parole-chiave **tabstave** viene utilizzata per creare una nuova riga della tablatura vuota. Per riempire la tablatura è necessario utilizzare la parole-chiave **notes**. Le note vengono rappresentate nella forma fret/string, ovvero tasto/corda. Per aggiungere più note sulla stessa corda si utilizza la dicitura notes fret-fret-fret/string.

Per aggiungere la barra di separazione delle battute viene utilizzato il carattere | . Si possono utilizzare un numero di barre di separazione a proprio piacimento.

Naturalmente per poter rappresentare in maniera completa una tablatura, è necessario predisporre un sistema per l'implementazione di bending, slide, vibrato, hammer-on e pull-off.

Per disegnare un bend, si separano due tasti dal carattere b. La differenza tra i due tasti determina l'entità del bend. Ad esempio 10b12 rappresenta un full-step-bend.

Aggiungendo il carattere v alla fine delle note si inserisce un vibrato, ad esempio 3-4v/2 aggiunge un vibrato sul 4 tasto della seconda corda.

Per rappresentare hammer-on, pull-off, taps o slide si usano i caratteri h, p, t oppure s, rispettivamente. Ad esempio, per rendere un hammer-on / pull-off dal tasto 6 all'8 e viceversa si usa 6h8p6.

6.3 Tab Div

Tab Div permette di incorporare le tablature per chitarra all'interno di blog, siti web, senza il bisogno di utilizzare applicazioni esterne. Ciò significa poter creare, editare e condividere le tablature semplicemente editando il codice scritto in Vex Tab all'interno di un semplice documento HTML.

Per incorporare le tablature bisogna eseguire il download di TabDiv dal sito web ufficiale di VexFlow – www.vexflow.com – ed estrarre i file contenuti all'interno del pacchetto compresso.
Dopodichè bisogna includere i sorgenti di VexFlow, TabDiv e jQuery (dalla versione 1.4 in poi) all'interno del documento HTML.
Ecco un esempio di come andrebbero inclusi i sorgenti:

```
Esempio 6.2
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4/jquery.min.js">
</script>
<script src="http://path/to/vexflow.js"></script>
<script src="http://path/to/vextabdiv.js"></script>
<link   href="http://path/to/tabdiv.css"   media="screen"   rel="Stylesheet"
type="text/css" />
```

Ogni elemento DIV che viene aggiunto al documento HTML, per rendere una tablatura per chitarra, deve contenere la classe vex-tabdiv.
Nel seguente esempio viene mostrato come riempire un DIV, di larghezza uguale a 400 pixel.

```
Esempio 6.3
<div class="vex-tabdiv" width=400 scale=0.8>
  tabstave
  notes (5/2.6/3.7/4) 5h6/3 7/4 | 5h7p5/4 7/5 5s3/5

  tabstave
  notes (8/2.7b9b7/3) (5b6/2.5b6/3)v 7s0/4 | (5/2.6/3.7/4)v
<div>
```

Per maggiori informazioni e per rimanere informati su gli sviluppi di VexFlow è possibile consultare il sito web ufficiale – www.vexflow.com – oppure collegarsi al blog del team di sviluppo - http://0xfe.blogspot.com/ - contenente numerose altre applicazioni di JavaScript in ambito musicale.

CAPITOLO 7

LILYPOND

7.1 Cos'è LilyPond

LilyPond è un software libero per la notazione musicale, disponibile per tutti i principali sistemi operativi. Utilizza una notazione testuale per la musica basata sull'insieme dei caratteri ASCII che viene poi compilata per ottenere documenti PDF, PostScript, SVG, PNG e altri formati. LilyPond inoltre può generare file MIDI.

A differenza di altri diffusi programmi commerciali, come Finale e Sibelius, LilyPond non possiede un'interfaccia grafica integrata per la creazione degli spartiti, ma ha comunque come obiettivo di produrre un output comparabile agli spartiti stampati professionalmente.

Uno dei principali vantaggi di LilyPond è quello di produrre spartiti di alta qualità, disegnati seguendo le regole di scrittura tradizionali, ispirandosi all'epoca in cui i punzoni di stampa degli spartiti erano posizionati a mano. Spesso gli spartiti prodotti con LilyPond vengono considerati migliori di quelli fatti con programmi commerciali, nonostante questi ultimi siano molto migliorati negli ultimi anni.

LilyPond è sviluppato da una comunità molto attiva: sono rilasciati continui aggiornamenti consistenti in risoluzioni di bug per la versione stabile, e in aggiunta di nuove funzionalità per la versione in sviluppo. Inoltre, essendo scritto in C++ ed assemblato mediante una libreria Scheme (GNU Guile), consente all'utente l'aggiunta di estensioni e personalizzazioni.

7.2 Compilare Musica

LilyPond è un sistema compilato: viene eseguito su un normale file di testo che descrive la musica. L’output risultante può essere visualizzato sullo schermo o stampato. In un certo senso, LilyPond è più simile a un linguaggio di programmazione che a un software grafico per la creazione di spartiti.

Non si scrive la musica trascinando le note da una barra degli strumenti grafica e mettendole su una partitura che si aggiorna in modo dinamico; si scrive musica digitando del testo. Questo testo viene interpretato (o “compilato”) da LilyPond, che produce degli spartiti di elevata qualità, esportabili in diversi formati.

Il file salvato, per poter essere compilato deve avere estensione .ly. Di seguito verranno presentati alcuni esempi di input testuale.

7.2.1 Il file di input

I file di input di LilyPond sono simili ai file sorgenti di molti comuni linguaggi di programmazione. Contengono una dichiarazione di versione, sono sensibili alle maiuscole, e in generale gli spazi bianchi vengono ignorati. Le espressioni musicali si formano con parentesi graffe { }, e i commenti sono indicati con % o %{ … %} .

Nella creazione di un file testuale LilyPond, bisogna specificare la dichiarazione di versione. Una dichiarazione di versione è una linea che indica la versione di LilyPond per la quale il file è stato scritto, come nel seguente esempio:

```
\version "2.14.2"
```

Per convenzione, la dichiarazione di versione viene posta all'inizio del file LilyPond.

La dichiarazione di versione è importante per almeno due ragioni. Primo, permette l'aggiornamento automatico del file di input file via via che la sintassi di LilyPond cambia. Secondo, indica la versione di LilyPond richiesta per compilare il file.

Se la dichiarazione di versione è omessa dal file di input, LilyPond mostra un avvertimento durante la compilazione del file.

LilyPond è Case-Sensitive, ovvero distingue tra lettere in minuscolo (es. a,b,c) o in maiuscolo (es. A,B,C). Se le note sono in minuscolo è un input valido, altrimenti causerà un messaggio di errore.

Inoltre, LilyPond è insensibile agli spazi, pertanto non importa quanti spazi (o tabulazioni o nuove linee) si aggiungono.

Una buona regola pratica è indentare i blocchi di codice con una tabulazione o due spazi:

```
{
  c4 d e
}
```

Ogni parte dell'input di LilyPond deve avere { parentesi graffe } intorno. Queste parentesi dicono a LilyPond che l'input costituisce un'espressione musicale singola, proprio come le parentesi () in matematica. Per evitare ambiguità, le parentesi dovrebbero essere racchiuse tra spazi, a meno che non si trovino all'inizio o alla fine di una linea.

Come in tutti i linguaggi di programmazione, è possibile inserire dei commenti all'interno del codice.

Un commento è un appunto per il lettore umano dell'input musicale che viene ignorato quando l'input viene analizzato, dunque non ha alcun effetto sull'output finale. Ci sono due tipi di commenti. Il simbolo di percentuale % introduce un commento di linea, per cui tutto quello che sta

dopo % su quella linea verrà ignorato. Per convenzione, un commento di linea viene posto sopra il codice a cui si riferisce.
Un commento di blocco segna come commento un'intera sezione di input musicale. Tutto ciò che è compreso tra %{ e %} viene ignorato. Tuttavia, i commenti di blocco non si 'annidano'. Ovvero non si può inserire un commento di blocco dentro a un altro commento di blocco. Provandoci, il primo %} interromperà entrambi i commenti di blocco.

Il seguente frammento di codice mostra gli usi possibili per i commenti:

```
% ecco le note di "Ah! Vous dirai-je, Maman"
  c4 c g' g a a g2

%{
  Questa linea e le note sotto vengono ignorate, perché si trovano in un commento di blocco.

  f4 f e e d d c2
%}
```

Un esempio basilare di un file di input di LilyPond.

```
\version "2.14.2"

\header { }

\score {
  ...espressione musicale composta...  % tutta la musica va scritta qui
  \layout { }
  \midi { }
}
```

7.2.2 Lo spartito

Nella composizione dello spartito, le note vengono codificate con lettere e numeri.

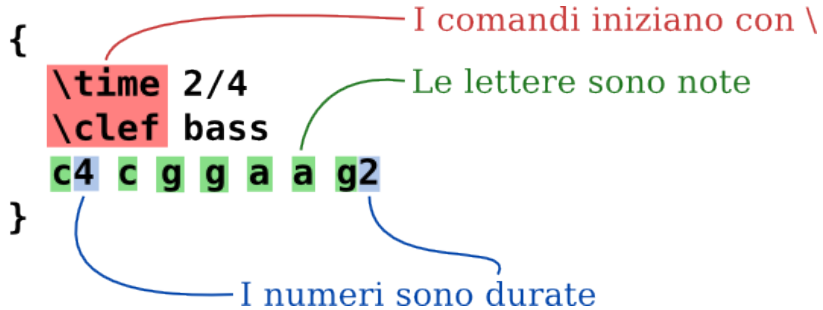


Figura 7.2.2.1 – Composizione spartito

In figura 7.2.2.1 notiamo che il tempo per ogni battuta è di 2\4, che la chiave è di Basso e che le note sono Do (durata 1\4), Do (durata 1\4), Sol, (durata 1\4) Sol (durata 1\4), La (durata 1\4), La (durata 1\4), Sol (durata 2\4).

Il risultato prodotto dal codice LilyPond è mostrato in figura 7.2.2.2.

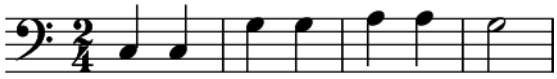


Figura 7.2.2.2 – Output codice LilyPond

Come un buon editor musicale, LilyPond permette di creare articolazioni, mostrare i nomi degli accordi e inserire parti per orchestra.

Di seguito alcuni esempi.

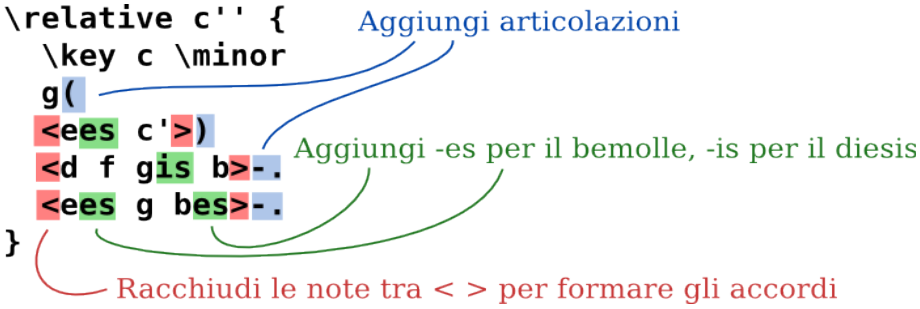


Figura 7.2.2.3 – Creazione articolazioni

Da notare che in LilyPond le alterazioni si producono scrivendo “es” per il bemolle e “is” per il diesis.

Il risultato prodotto dal codice mostrato in Figura 7.2.2.3 è il seguente:

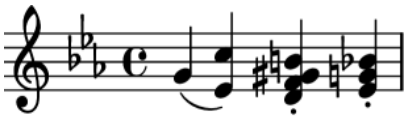


Figura 7.2.2.4 – Output articolazioni

Nell'esempio seguente vengono inseriti, sopra la notazione musicale inserita nello spartito, i nomi degli accordi che serviranno da accompagnamento alla melodia.

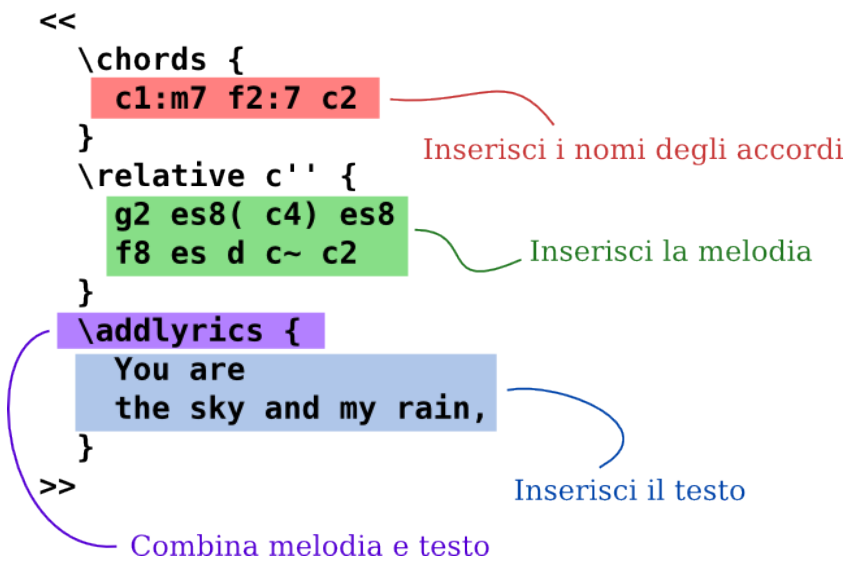


Figura 7.2.2.5 – Creazione accordi

All'interno del codice, si nota il paradigma "\chords {}" che contiene la sintassi per la generazione dell'accordo.

Ecco il risultato prodotto dal codice mostrato in figura 7.2.2.5

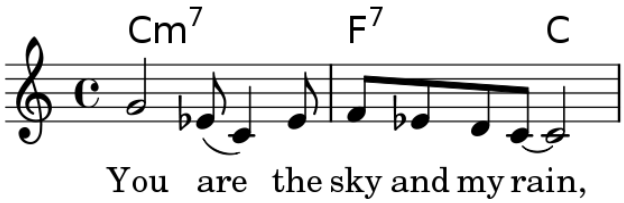


Figura 7.2.2.6 – Visualizzazione accordi

7.3 Creare tablature con LilyPond

LilyPond oltre che rappresentare un potente linguaggio di programmazione per la generazione di spartiti professionali, supporta anche la notazione per tablature che può essere personalizzata per adattarsi a qualsiasi strumento che sia in grado di leggere le intavolature.

Generare tablature, scritte in LilyPond è davvero semplice, basta seguire la sintassi.

Il numero di corda associato ad una nota viene rappresentato da un backslash seguito da un numero. Di default, la corda 1 è la più acuta, mentre l'accordatura utilizzata è quella standard per chitarre a sei corde (E A D G B E).

Ecco un semplice esempio di tablatura per chitarra:

```
\new TabStaff {
  a,4\5 c'\2 a\3 e'\1
  e\4 c'\2 a\3 e'\1
}
```

Il tag \new TabStaff avvisa LilyPond che il codice contenuto tra le parentesi graffe si riferisce ad una tablatura.
Il risultato generato dal codice menzionato poc'anzi è il seguente:

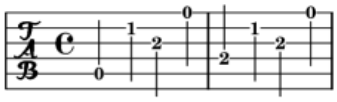


Figura 7.3.1 – Tablatura generata in LilyPond

Analizzando il codice si nota che non vengono inseriti il numero dei tasti da visualizzare sulla corda. LilyPond in maniera automatica produce la tablatura considerando soltanto il nome della nota, l'ottava della nota e la corda.
Per riconoscere l'ottava corrispondente alla nota da rappresentare, vengono implementati i seguenti simboli:

- “ , ” per la terza ottava
- nessun simbolo per la quarta ottava
- “ ' “ per la quinta ottava
- “ '' “ per la sesta ottava
- “ ''' “ per la settima ottava
- “ '''' “ per l'ottava ottava

Risulta quindi semplice scrivere sia tablature che spartiti con LilyPond, inoltre, essendo gratuito e libero, ogni utente ha la possibilità di poter implementarne il codice sorgente, creare nuove funzionalità ed in caso di dubbi è disponibile un'ampia documentazione sul sito web ufficiale del progetto.

CAPITOLO 8

IEEE 1599 E TABLATURE: L'APPLICAZIONE WEB

8.1 Introduzione

Come già menzionato nel capitolo 2 del presente elaborato, lo standard IEEE 1599 è un formato XML con struttura multi-layer che permette la rappresentazione di elementi musicali eterogenei.

Lo sviluppo e gli studi su IEEE 1599 sono stati eseguiti presso il laboratorio di Informatica Musicale (LIM) dell'Università degli Studi di Milano, sito in via Comelico 39.

Il Laboratorio d'Informatica Musicale, attivo dal 1985 sotto la direzione del prof. Goffredo Haus, è uno dei principali laboratori di ricerca del dipartimento di Informatica e Comunicazione dell'Università degli Studi di Milano. Nel corso degli anni ha partecipato a numerosi progetti di ricerca a livello mondiale, realizzando progetti applicativi per il Teatro alla Scala di Milano, RAI Radiotelevisione Italiana, Teatro Bolshoi di Mosca, RSI Radiotelevisione Svizzera italiana, Orchestra Verdi di Milano, IEEE Computer Society, Charles Ames Research Center della NASA, l'Archivio Storico Ricordi, Consorzio Nazionale delle Ricerche (CNR) e Ministero per i Beni e le Attività Culturali. Ai già citati importanti progetti si aggiungono numerose collaborazioni con case discografiche, case editrici, studi di produzione musicale e radiotelevisiva, industrie di strumenti musicali, software house e internet companies.

Nell'ottica degli studi su IEEE 1599 ricade il presente elaborato finale, volto sia ad un'attenta analisi dello standard prodotto dal LIM e ad un suo possibile utilizzo sul Web.

L'obiettivo primario è poter rendere semplice la conversione di uno spartito, esportato in formato XML IEEE 1599, in tablatura per chitarra.

Esistono numerosi software di notazione musicale e di intavolature per strumenti a corda con i quali è possibile leggere documenti XML derivanti da partiture. Si tratta comunque di software che necessitano naturalmente di un'installazione sulla propria macchina e che occupano una buona porzione di memoria per la loro esecuzione.

Oggigiorno il Web ha raggiunto traguardi notevoli, è entrato nelle case delle persone ed ha raggiunto ognuno di noi. E' possibile navigare sul Web da qualsiasi macchina che disponga di hardware ethernet/wifi, se si è stabilito un contratto con una compagnia telefonica, oppure da qualsiasi dispositivo mobile che permetta la navigazione Wap o Web.

La potenza del Web, la facilità con il quale è possibile reperire informazioni, rispecchia gli obiettivi, seppur didattici, di questo elaborato:

Rendere semplice, rapida, la conversione dell'XML IEEE 1599 in tablatura per chitarra.

8.2 Panoramica sul progetto

Per lo sviluppo del progetto è stato scelto l'ambiente di lavoro ASP.NET MVC con Microsoft Visual Studio 2010. La scelta di sviluppare il progetto secondo tale linguaggio è stata presa, primo perchè l'ambiente .NET mette a disposizione una serie di tools potenti per lo sviluppo di applicazioni web, sia perchè l'utilizzo congiunto del pattern MVC e della tecnologia LINQ, rende possibile e semplice lo sviluppo di quelle applicazioni per cui, in altri linguaggi, si necessiterebbe dell'inclusione di numerose librerie e della scrittura di codice notevole.

Nel capitolo 4 è stato spiegato il significato di ASP.NET MVC, della sua struttura e del suo funzionamento, si rimanda a tale capitolo per informazioni sul linguaggio.

La programmazione orientata agli oggetti (OOP) permette di modellare l'idea di un progetto come costituito da uno o più oggetti, che eseguono una serie di operazioni e che si scambiano dati.

Nel progetto si è pensato, da un punto di vista teorico, di vedere il tutto come un unico oggetto Tablatura, contenente a sua volta diversi oggetti, ovvero, costituito dall'oggetto nota, contenente le informazioni sulle note dello spartito, l'oggetto accordo, contenente gli oggetti note costituenti l'accordo e l'oggetto misura, contenente gli oggetti accordo costituenti una battuta.

L'insieme degli oggetti misure va a costituire l'oggetto Tablatura, ovvero il nostro risultato finale.

Detto in così poche righe, sembrerebbe semplice sviluppare l'applicazione, considerato che si tratta di un modello box in a box, una sorta di Lego, dove mattoncino sopra mattoncino si va costruendo un piccolo edificio.

E' stato specificato il punto di vista teorico di come andrebbe sviluppata l'applicazione, di seguito verranno specificate le scelte di programmazione utilizzate.

8.3 Analisi delle tecnologie utilizzate

Il punto di partenza nell'analisi del progetto è stato lo studio del formato XML IEEE 1599, degli elementi che lo costituiscono ed in particolare di quegli elementi contenenti le informazioni musicali da rappresentare su tablatura.

L'analisi è ricaduta all'interno del layer Logic e del suo sottoelemento LOS, contenente per l'appunto tutti gli elementi essenziali per lo studio della struttura musicale della partitura.

Il LOS si costituisce dei seguenti sotto-elementi:

- staff_list
- part

Lo **staff_list** contiene informazioni sull'accollatura, ossia l'elenco degli staff, contenute nel suo sotto-elemento staff. Lo staff è il pentagramma e contiene attributi quali line_number, che identificano il numero di righe dello spartito, ed id, che identifica lo staff appartenente ad una determinata Parte.
All'interno dello staff troviamo due ulteriori sottoelementi, il time_signature, contenente informazioni temporali, ed il key_signature, rappresentante informazioni sulla chiave della Parte a cui lo staff_list si riferisce.
Di seguito un esempio di staff_list:

```
Esempio 7.1
<staff_list>
      <staff id="part_1_staff" line_number="5">
            <time_signature event_ref="TimeSignature_part_1_1">
                  <time_indication num="4" den="4"/>
            </time_signature>
            <key_signature event_ref="KeySignature_part_1_1">
                  <sharp_num number="0"/>
            </key_signature>
            <clef shape="G" event_ref="Clef_part_1_1" staff_step="2"
            octave_num="0"/>
      </staff>
</staff_list>
```

L'elemento Part, invece, contiene tutte le informazioni inerenti le voci della partitura.
Al suo interno troviamo il sotto-elemento **measure**, avente attributo number, per la sua identificazione, che presenta tutte le informazioni riguardanti le note musicali esistenti in quella determinata misura o battuta. Sono presenti così gli elementi **chord**, contenenti a loro volta gli empty tag duration, che determinano la durata delle note, e l'elemento **notehead** che contiene l'empty tag pitch. Il pitch rappresenta un elemento fondamentale, in quanto contiene le informazioni di rappresentazione fondamentali della nota, ovvero, l'attributo octave, che individua l'ottava della nota, l'attributo step, che definisce quale sia la nota, ed infine l'attributo actual_accidental, che definisce se la nota sia naturale, bemolle o diesis.
Di seguito un esempio rappresentante un elemento part.

```
Esempio 7.2
<part id=”part_1”>
      <voice_list>
            <voice_item id="part_1_0_voice" staff_ref="part_1_staff"/>
      </voice_list>
      <measure number="1">
            <voice voice_item_ref="part_1_0_voice">
                  <chord event_ref="part_1_voice0_measure1_ev0">
                        <duration num="1" den="8"/>
                        <notehead>
                              <pitch octave="5" step="C"
                              actual_accidental="natural"/>
                        </notehead>
                  </chord>
            </voice>
      </measure>
</part>
```

Considerato che il nostro scopo è rappresentare uno spartito, esportato in XML IEEE 1599, in tablatura per chitarra, tutti gli altri elementi costituenti il nostro file verranno scartati, in quanto non utili nel raggiungimento dello scopo finale.

Superata la fase d'analisi dello standard IEEE 1599 è necessario porsi la domanda su come si possano estrarre i dati dal file, in modo tale da poter essere elaborati dall'applicazione.

La soluzione ricade sulla tecnologia LINQ, Language-Integrated Query, un set di funzionalità introdotto in Visual Studio 2008 che migliora la gestione delle query nella sintassi dei linguaggi C# e Visual Basic. Visual Studio include degli assembly del provider LINQ che consentono l'utilizzo di LINQ con insiemi di .NET framework, database SQL Server, DataSet ADO.NET e soprattutto documenti XML.

Nello sviluppo dell'applicazione, in modo particolare, è stato utilizzato LINQ to XML, che fornisce un'interfaccia di programmazione ed una serie di tool per la creazione, la manipolazione di file XML e l'esecuzione di query su di essi.

Per poter utilizzare le funzionalità di LINQ to XML è necessario includere la direttiva assembly using System.Linq e using System.Xml.Linq.

Dall'integrazione di LINQ, XML IEEE 1599 e ASP.NET MVC è nata l'applicazione per la creazione di tablature a partire da XML secondo lo standard 1599.

8.4 La struttura dell'applicazione

Come già accennato poc'anzi, si è pensato di lavorare su una struttura gerarchica, mattone su mattone, che andasse a creare la struttura finale, costituita dalla tablatura ottenuta dall'elaborazione del file XML dato in input.

La procedura, la navigazione dell'applicazione, avviene in maniera molto semplice.

Nella homepage, in alto a destra, sono presenti due link, Home e Upload, il primo rimanda l'utente alla homepage, mentre il secondo alla pagina di upload dei file.

E' stato predisposto un sistema di registrazione e autenticazione degli utenti, pertanto per poter caricare un determinato file sul server, è necessario autenticarsi. E' stato implementato un sistema che ad autenticazione non avvenuta, nel caso di click su “Upload”, rimanda l'utente alla pagina per il Login.

Il processo di registrazione è molto semplice e si basa su un form contenente alcuni campi da compilare, essi sono il campo Username, Email address, Password e Confirm Password.

Il primo campo indica l'username che l'utente sceglierà per eseguire il Login al sito, il campo Email address indica l'indirizzo email dell'utente, mentre Password e Confirm Password richiedono la scelta di una password per l'autenticazione. Per motivi di sicurezza è stato scelto di non accettare password aventi lunghezza inferiore ai 6 caratteri.

Naturalmente, per eseguire correttamente la registrazione, tutti i campi devono essere compilati e sia Password che Confirm Password devono contenere la stessa sequenza di caratteri.

Effettuata la registrazione, l'utente può procedere con l'autenticazione, cliccando o su Upload, oppure su Log On, posto subito sopra il link Upload.

Al click su Log On, si apre la pagina di autenticazione, dove è richiesto che vengano immessi i dati inerenti Username e password. Nel caso di riscontro positivo tra i dati inseriti dall'utente e la ricerca dei dati all'interno del database, l'utente sarà autenticato e potrà procedere con l'upload del file XML.

Per eseguire l'upload basta cliccare sul pulsante “scegli file”, selezionare il file desiderato, e cliccare su “invia”.

E' proprio in questo momento che viene messa in moto l'intera macchina d'elaborazione.

In quanto non è possibile stabilire se l'utente, nel proprio file XML, abbia stabilito delle diciture standard per nominare un determinato elemento part, come ad esempio guitar o chitarra, viene visualizzata una tabella, creata dinamicamente a seconda del file caricato, che presenta l'elenco delle parti contenute all'interno del file XML. In questo modo, l'utente potrà cliccare sulla parte interessata, ottenendone la tablatura.

Ottenuta la tablatura all'interno della pagina Web, ritenendo di fondamentale importanza che l'utente possa salvare il proprio risultato, è stato implementato un sistema di creazione e download della tablatura sia in formato PDF che XML IEEE1599. In tal modo l'utente può scegliere se salvare il risultato ottenuto secondo i due formati citati.
Ritornando all'upload dei file, è stato implementato un sistema di controllo del caricamento. Esso prevede che si possano caricare solamente file aventi estensione .xml, in caso contrario viene visualizzata una schermata d'errore che avvisa l'utente che il file caricato non è di tipo XML.
Dopo aver spiegato in linea teorica la struttura ed il funzionamento dell'applicazione Web, è giunto il momento di visionare e capire, passo per passo, ogni elemento, parte, del progetto.

Nella figura sottostante, 8.1, viene visualizzata la Homepage. Si nota il menu con i link Home e Upload.

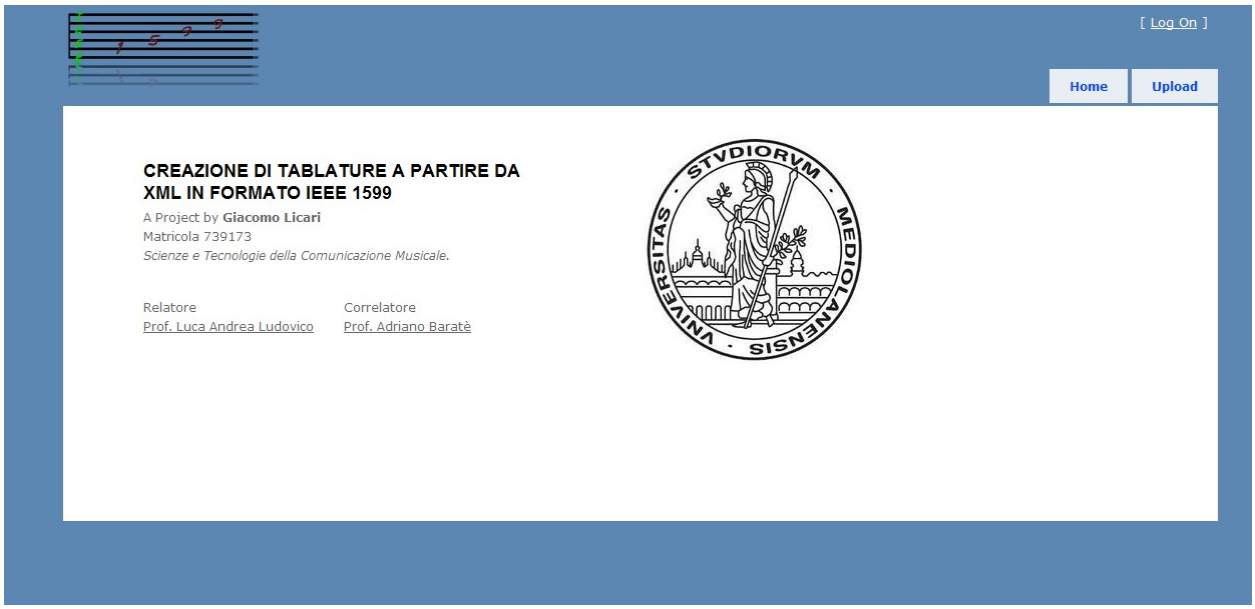


Figura 8.1 – Homepage

Nella figura 8.2 si evidenzia la pagina per il login, con il relativo form.

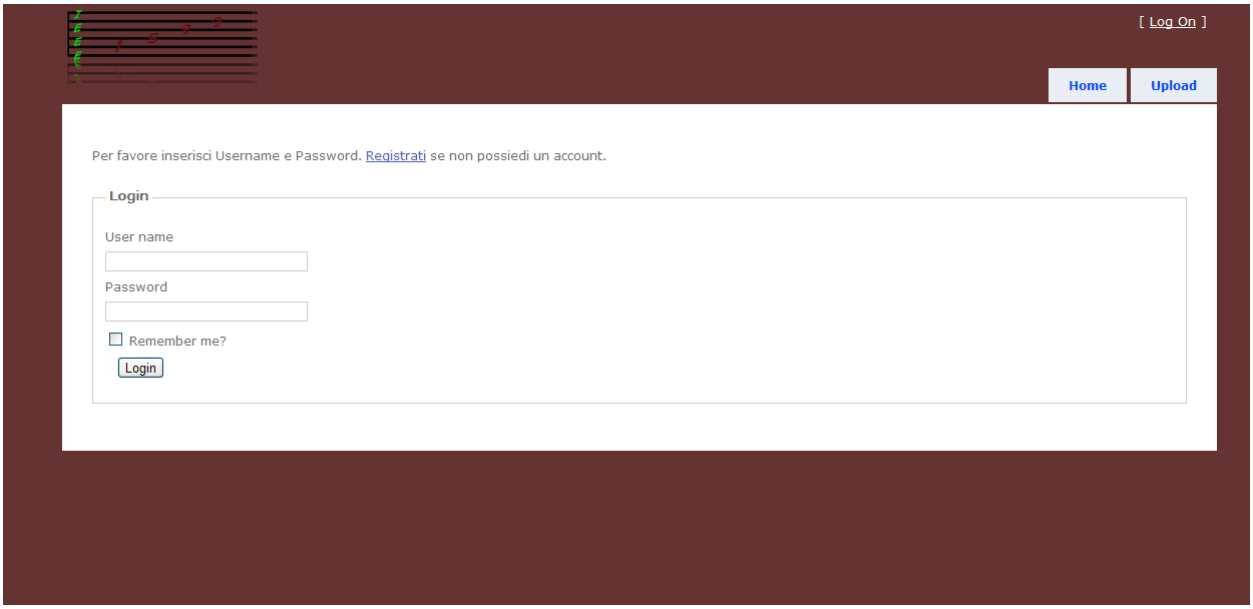


Figura 8.2 - Login

Nella figura 8.3 viene mostrata la pagina per l'upload dei file XML.

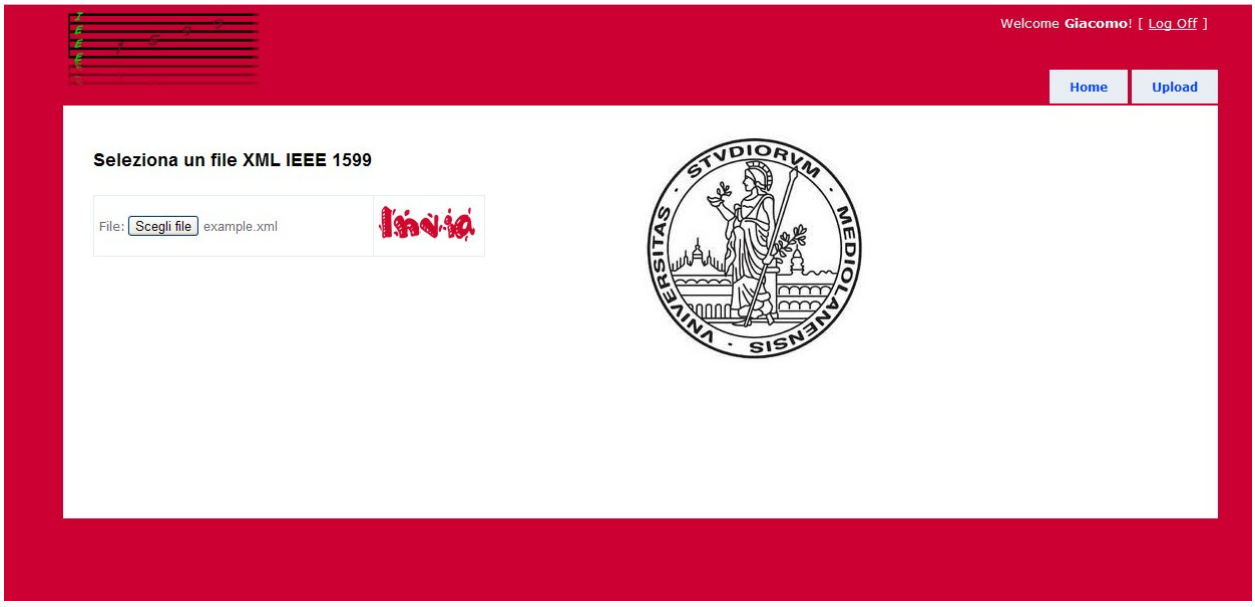


Figura 8.3 - Upload

Nella figura 8.4 l'utente accede alla pagina in cui viene generata la lista delle voci che compongono il file XML, nel caso dell'esempio è presente la sola voce Chitarra_1.

Figura 8.4 – Selezione Voce

La figura 8.5 mostra la tablatura finale ottenuta dalla traduzione di una partitura esportata in XML IEEE 1599. Nel caso viene presentata una scala di DO Maggiore.

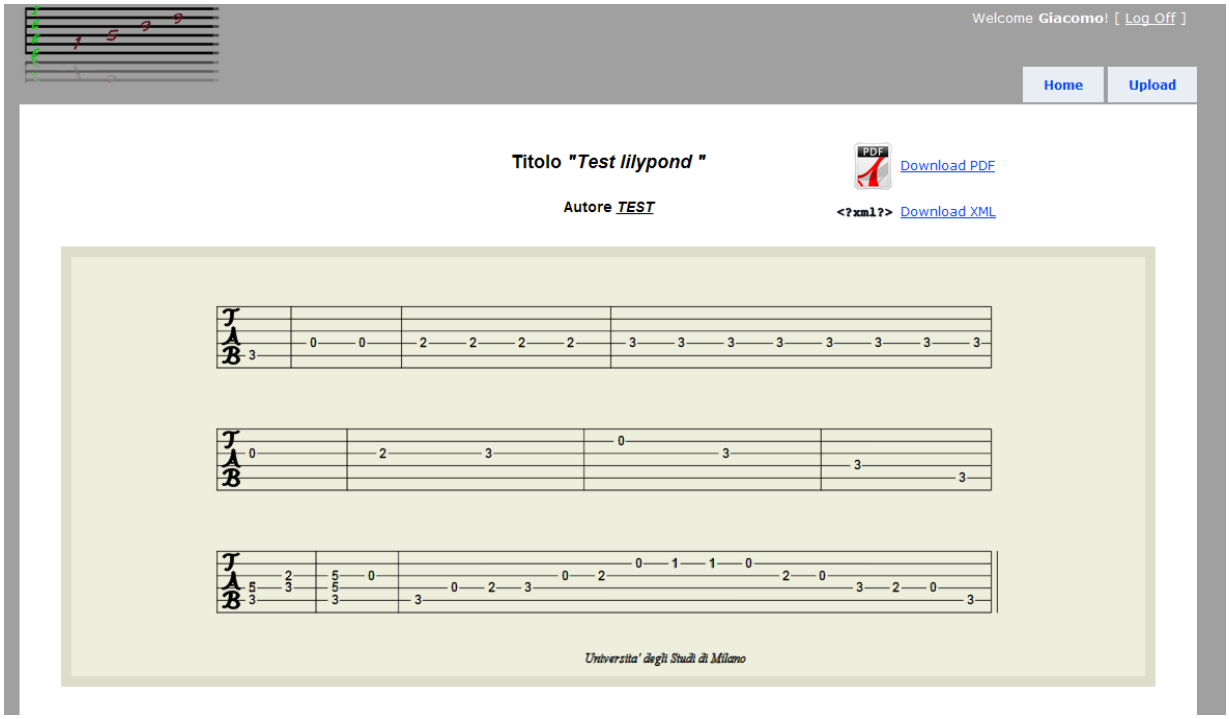


Figura 8.5 – La tablatura

Nella figura sottostante è rappresentato il logo del progetto, composto dalla struttura di una tablatura, contenente al posto della dicitura TAB, la dicitura IEEE, e al posto dei numeri rappresentanti le note, i numeri 1599, che si riferiscono allo standard XML studiato.

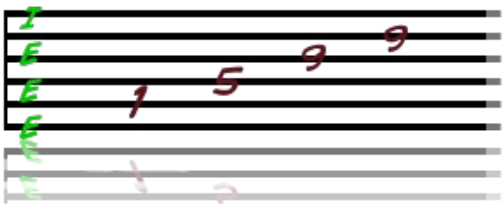


Figura 8.6 – Logo del progetto

8.5 Il Code-Behind

In questo capitolo verranno esposte parti di codice, volte a presentare l'architettura ed il codice del progetto, inoltre verrà presentata l'area di lavoro.

Quando si intende creare un'applicazione Web con Visual Studio 2010, bisogna cliccare su File > New > Project, aprendo così una schermata in cui sono presenti tutte le tipologie di applicazioni e servizi implementabili in ambiente .NET.

Per utilizzare il pattern MVC, si filtrano i risultati della schermata andando su Visual C# > Web e selezionando ASP.NET MVC 2 Web Application.

Selezionato il servizio da noi desiderato, Visual Studio apre l'area di lavoro in cui verrà sviluppata l'applicazione Web.

Si nota che sulla parte sinistra del monitor è posta la Solution Explorer, di default Visual Studio la pone in quell'area. Si tratta di un'area contenente tutte le cartelle i file del nostro progetto.

Alla base della Solution Explorer sono posti diversi tab, ognuno dei quali al click su di essi, punta ad un servizio diverso di Visual Studio, come ad esempio il Server Explorer, che gestisce tutte le connessioni del progetto a base di dati.

Tornando alla Solution Explorer è da notare che avendo scelto di sviluppare un'applicazione in ASP.NET MVC, sono presenti tre cartelle fondamentali: Controllers, Models, Views.

La cartella Controllers contiene tutti i controller necessari all'elaborazione dei dati e alla gestione delle azioni dell'applicazione. Per aggiungere un controller basta cliccare col tasto destro del mouse sulla cartella Controllers, selezionare Add > Controller, scegliere il nome da dare al controller, nella forma NomeController e cliccare ok.

La cartella Models contiene, invece, tutte le classi, le pagine, ma anche gli ADO.NET Data Model che costituiscono il modello dati dell'applicazione. Per aggiungere un item alla cartella Models si fa

tasto destro del mouse sulla cartella, Add > New Item, e si seleziona l'item che contribuirà alla creazione del modello dati.

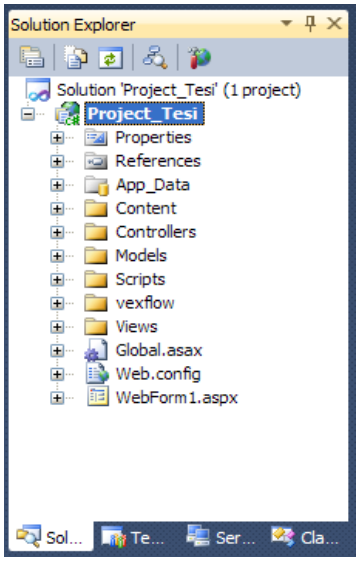
Infine è presente la cartella Views, contenente tutte le viste relative alle azioni gestite dai controller.

Per aggiungere una vista è necessario aprire una classe controller, posizionarsi sul codice inerente un metodo e cliccarvi sopra col tasto destro del mouse, selezionando Add View.

Una volta creata la Vista per quel determinato metodo, ogni azione che dovesse coinvolgere quel metodo genererà automaticamente del markup all'interno della vista ad esso collegata.

Di seguito è rappresentata la Solution Explorer del progetto di tesi.

Si notano le cartelle Controllers, Models e Views, oltre che le cartelle Content, contenente il file CSS del progetto e le immagini utilizzate nel progetto, Scripts, contenente gli script adottati di default da Visual Studio, più gli ulteriori script inseriti manualmente durante lo sviluppo del progetto, più la cartella vexflow che contiene i file per l'implementazione della visualizzazione finale della tablatura.



8.5.1 I controller

I controller utilizzati dall'applicazione sono diversi e riguardano, come detto precedentemente, ogni azione eseguibile dall'utente, come il login, la registrazione, l'upload e la scelta della parte di spartito da tradurre.

Essi sono, in ordine, l'AccountController, l'ErrorController, l'HomeController, il TablaturaController ed il VociController.

L'AccountController gestisce il Login, il Logout, la registrazione ed il cambio password dell'utente. Esso contiene i metodi LogOn (), LogOff (), Register (), ChangePassword (), per l'esecuzione delle

operazioni di login, logout, registrazione e cambio password, includendo all'interno di ogni metodo i controlli e le operazioni necessarie a soddisfare l'azione dell'utente.

L'ErrorController contiene poco codice, ma di fondamentale importanza nella gestione delle eccezioni. Quando durante l'esecuzione dell'applicazione si verifica un errore, il controller lo notifica all'utente, specificando inoltre la natura dell'errore.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Project_Tesi.Controllers
{
    public class ErrorController : Controller
    {
       public ActionResult Index()
        {
            Exception ex = null;

            ex =
(Exception)HttpContext.Application[Request.UserHostAddress.ToString()];

            ViewData["Title"] = "Siamo spiacenti, si è verificato un errore.";
            ViewData["Description"] = ex.Message;

            return View("Error");
        }
    }
}
```

La collection ViewData[“Title”] informa l'utente che si è verificato un errore, mentre ViewData[“Description”] riporta la natura dell'errore.

L'HomeController anch'esso è costituito da una manciata di codice necessario solamente a richiamare la vista Home quando si accede alla homepage.

Il VociController viene richiamato nel momento in cui si esegue l'upload del file XML.

Al file viene dato un nome univoco, derivante dalla creazione di una sessione apposita, e viene salvato nella cartella ~/App_Data/uploads/ del progetto.

Questo controller ha il compito di estrarre dal file caricato l'elenco delle parti che lo compongono e di creare dinamicamente una tabella, contenente l'elenco delle parti. Ogni parte costituisce un link, creato per mezzo degli HTML Helpers di Visual Studio e che consente di specificare il nome del link, la vista a cui fa riferimento il link, il controller che gestirà il dato ed infine permette di scegliere un id per quella stringa. In questo modo, cliccando sul nome della parte, il valore verrà inviato ad un secondo controller, il TablaturaController.

8.5.2 Il TablaturaController

Il TablaturaController rappresenta il cuore dell'intera applicazione, in quanto è all'interno di esso che viene applicata tutta la logica applicativa e le query che producono la tablatura finale.

In esso sono presenti tre metodi, il metodo Tablatura (string id, string sessione), che riceve dalla vista generata dal VociController, l'id, ovvero la stringa inerente la parte di spartito da tradurre e la stringa contenente il valore assegnato alla sessione, il metodo Download_PDF, che viene invocato quando un utente sceglie di eseguire il download del file PDF ottenuto dalla generazione della tablatura, ed infine il metodo Download_XML, che viene richiamato quando l'utente richiede di eseguire il download del file tablatura in formato XML IEEE1599.

Innanzitutto, prima di passare alla creazione della nostra tablatura, seppur lieve, è stato implementato un sistema di controllo della voce selezionata dall'utente. Lieve, in quanto non è possibile stabilire quale dicitura l'utente abbia utilizzato nella creazione del proprio spartito per riferirsi allo strumento Chitarra. Il controllo prevede l'analisi delle ottave presenti nel file XML, sapendo che la chitarra comprende note che vanno dalla terza all'ottava ottava. Nel caso venissero riscontrate note al di fuori di tale range, viene segnalata all'utente la possibilità che la voce selezionata possa non riferirsi ad una chitarra e che pertanto potrebbero verificarsi degli errori di traduzione. Nella pagina seguente viene mostrato il codice implementato per il controllo delle ottave presenti all'interno dello spartito.

```
                XDocument doc_intero = XDocument.Load(Request.MapPath("~/App_Data/uploads/" +
sessione_tab));

            //////////////////////
            // Controllo Parte //
            //////////////////////
            IEnumerable<XElement> control_part = from c in doc_intero.Root.Descendants("part")
                                                 where (string)c.Attribute("id") == id_part
                                                 select c;

            foreach (XElement x in control_part)
            {
                IEnumerable<XElement> control_measure = from c in
control_part.Descendants("measure")
                                                        select c;

                foreach (XElement misura_c in control_measure)
                {
                    IEnumerable<XElement> control_chord = from c in
misura_c.Descendants("chord")
                                                          select c;

                    foreach (XElement chord_c in control_chord)
                    {
                        IEnumerable<XElement> control_pitch = from c in
chord_c.Descendants("pitch")
                                                              select c;
                        foreach (XElement pitch_c in control_pitch)
                        {
                            switch ((int)pitch_c.Attribute("octave"))
                            {
                                case 1:
                                    ViewData["Controllo"] = "La partitura potrebbe non essere
per chitarra, pertanto potrebbero visualizzarsi degli errori di traduzione.";
                                    break;
                                case 2:
                                    ViewData["Controllo"] = "La partitura potrebbe non essere
per chitarra, pertanto potrebbero visualizzarsi degli errori di traduzione.";
                                    break;
                                case 8:
                                    ViewData["Controllo"] = "La partitura potrebbe non essere
per chitarra, pertanto potrebbero visualizzarsi degli errori di traduzione.";
                                    break;
                                default:
                                    break;
                            }
                        }
                    }
                }
            }
```

All'interno del TablaturaController vengono eseguite una serie di query volte ad estrarre dal file XML la parte di spartito interessata, le misure presenti nella parte, i chords contenuti all'interno delle misure e le note contenute all'interno dei chords. Si parla di chords in quanto lo standard IEEE 1599 prevede che seppur allo stesso tempo su più righi esista una sola nota, questa viene considerata come un accordo.

Analizzata la parte relativa al controllo sulle note ed ottave presenti nello spartito, inizia la parte di analisi, estrazione e creazione dell'oggetto tablatura.

Per la visione completa del codice relativo ai Controller si rimanda all'Appendice.

Come già accennato ad inizio capitolo, è stato implementato un sistema di creazione e download della tablatura sia in formato PDF che in formato XML IEEE 1599.

Per quanto riguarda il formato PDF, in quanto non è possibile sapere a priori quante battute possa contenere la tablatura, è stata esclusa la pista relativa alla creazione di un template statico per la generazione del PDF. La scelta è ricaduta su LilyPond, un software libero per la notazione musicale, disponibile per tutti i principali sistemi operativi. LilyPond Utilizza una notazione testuale per la musica basata sull'insieme dei caratteri ASCII che viene poi compilata per ottenere documenti PDF, PostScript, SVG, PNG e altri formati.

I documenti ottenuti in LilyPond sono paragonabili con documenti ottenuti tramite software di notazione musicale tradizionali, pertanto grazie ad esso, il sistema mette a disposizione dell'utente la possibilità di eseguire il download della tablatura in formato PDF stampabile.

Di seguito vengono mostrate sia la versione Web che la versione PDF della tablatura.

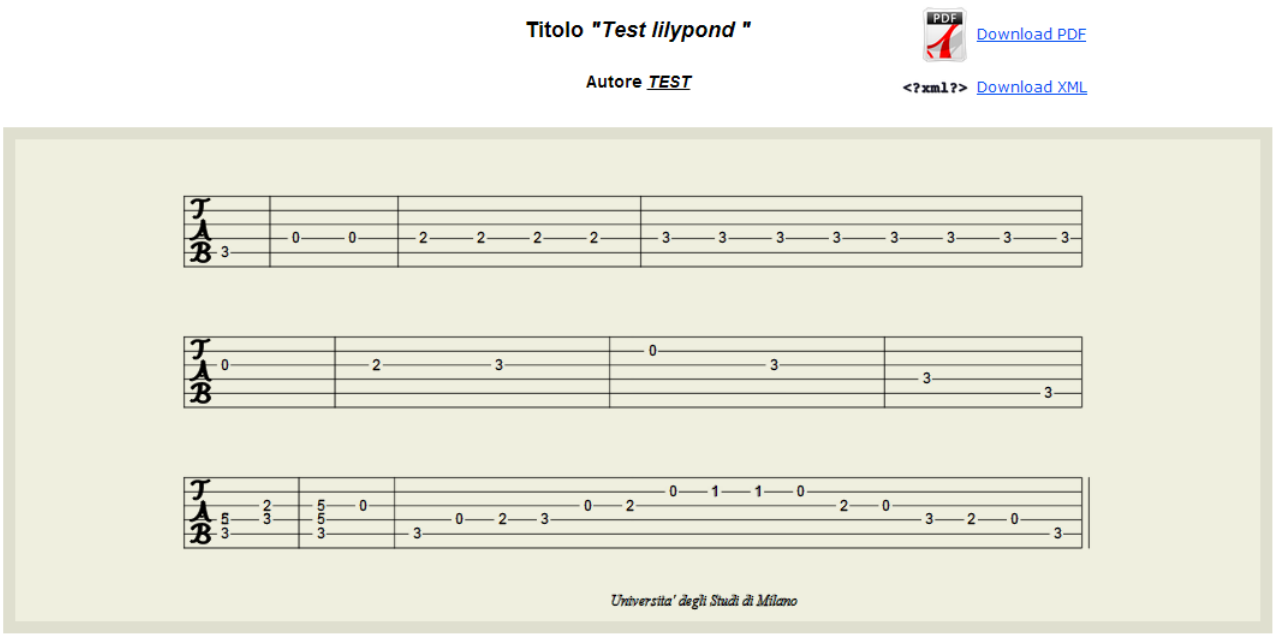


Figura 8.7 – Tablatura versione Web

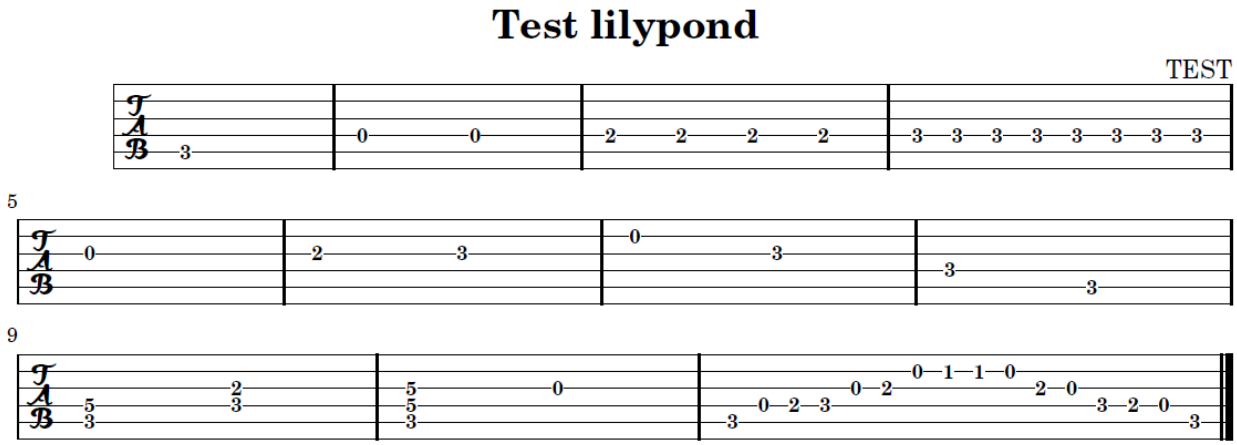


Figura 8.8 – Tablatura versione PDF

Oltre al file PDF è possibile effettuare il download dell'XML IEEE 1599. In questo caso viene creato un nuovo file XML che assume il nome di TAB + sessione.xml e che contiene di default lo stesso contenuto del file originario caricato dall'utente.

In questo file, tramite Linq to XML, vengono generati i tag inerenti la tablatura, secondo il DTD, e popolati con i dati contenuti all'interno delle varie classi che compongono il progetto.

In questo modo viene servito all'utente uno strumento completo per la traduzione ed il salvataggio della tablatura secondo il formato desiderato.

8.5.3 I Models

Sono stati presentati i controller che costituiscono l'ossatura dell'applicazione Web, di seguito verranno presentati i modelli e le viste prodotte.

Nella cartella Models sono presenti le seguenti classi: accordo, chitarra, global, misura, nota, tablatura e voci.

Partiamo, nell'analisi dei modelli, dalla classe che rappresenta l'elemento base della tablatura, ovvero la classe nota.

La classe nota contiene, in relazione alla struttura dell'empty tag contenuta in notehead, un int ottava, una string step, un elemento accidental di tipo accidental, derivante quindi dall'enumerazione accidental contenuta nella global, un int tasto ed un int corda.

Sono stato inseriti i due interi tasto e corda per determinare, come verrà spiegato più avanti, la posizione della nota sul manico della chitarra.

La classe accordo, contiene una lista chiamata note, fortemente tipizzata e di tipo nota. Essa serve al TablaturaController per aggiungere alla lista ogni nuova nota che viene riscontrata durante il parsing del documento XML.
A sua volta, la classe misura, contiene una lista fortemente tipizzata e di tipo accordo, chiamata accordi, che contiene l'insieme degli accordi che formano una misura. Anch'essa viene istanziata dal TablaturaController durante il parsing del documento.
La classe voci, contiene una lista di stringhe, denominata VociList e che viene popolata, dall'estrazione dal documento XML, con le parti che compongono lo spartito. Tale classe viene istanziata dal VociController.
La classe global, contiene un'enumerazione e due strutture struct.
L'enumerazione è denominata accidental e rappresenta la natura della nota, flat, natural, sharp, ponendoli rispettivamente uguali a -1, 0, 1.
Le due struct invece sono posizione e pitch.
Posizione contiene due interi, tasto e corda, mentre pitch contiene un int ottava, e le stringhe step e accidental.
La classe chitarra è molto importante e presenta due metodi, il metodo Chitarra () ed il metodo get_posizione (List<nota>note, int ottava, string nota, int alterazione) di tipo posizione, quindi derivante dalla struct posizione.
Il metodo chitarra () carica il file XML chiamato chitarra.xml, presente all'interno della cartella App_Data, che contiene le ottave rappresentabili sulla chitarra. Le ottave vanno dalla terza alle settima ed in particolare la terza ottava inizia con la nota MI, mentre la settima finisce con la nota RE. La struttura di chitarra.xml è molto semplice e presenta il root-tag chitarra con all'interno gli elementi ottava, aventi attributo id per identificarli. All'interno di ogni ottava sono presenti gli elementi nota, con l'id che identifica la nota (es. E) la cui composizione è descritta di seguito.

```
<nota id="E">
      <alterazione id="0">
            <posizione id="1">
          <corda>6</corda>
          <tasto>0</tasto>
            </posizione>
      </alterazione>
      <alterazione id="1">
            <posizione id="1">
          <corda>6</corda>
          <tasto>1</tasto>
            </posizione>
      </alterazione>
    </nota>
```

Essendo il MI la nota di partenza della terza ottava, non presenta l'alterazione con id uguale a -1.
Il metodo get_posizione di tipo posizione, viene istanziato dal TablaturaController e serve ad eseguire le query di estrazione di ottave, note, alterazioni e posizione delle note (tasto/corda) da chitarra.xml in base ai valori dati in input al metodo dal controller. Il metodo eseguirà il parsing di chitarra.xml col fine di individuare il tasto e la corda di ogni nota presente nel file XML caricato dall'utente.
Infine la classe tablatura serve ad aggiungere ad una lista, fortemente tipizzata e di tipo misura, l'intero oggetto tablatura prodotto dal TablaturaController.
Contiene inoltre due stringhe, titolo ed autore, per passare alla vista prodotta il titolo e l'autore della partitura da visualizzare all'utente.
L'insieme di tutte le classi menzionate fin'ora, contenute all'interno della cartella Models, vanno a costituire il ViewData.Model, richiamato dalle viste per confrontarsi col modello dati e che servirà per la modellazione dell'oggetto finale restituito all'utente.

8.5.4 Le Views

Le viste risiedono nella cartella Views del progetto che sono Account, Home, Shared, Tablatura, Voci.
La cartella Account genera le viste ChangePassword.aspx, LogOn.aspx, Register.aspx, per le operazioni di cambio password, login e registrazione degli utenti. E' da notare l'estensione .aspx che identifica le pagine create in ASP.NET.
La cartella Home contiene soltanto la vista index.aspx, responsabile della generazione del markup per la visualizzazione della Homepage, gestita dall'HomeController.
La cartella Shared contiene le viste Error.aspx ed i file Site.Master e LogOnUserControl.ascx, la prima genera il markup contenente eventuali errori da notificare, mentre il secondo file setta la struttura del markup base HTML da cui le viste si genereranno mentre il terzo mostra un messaggio di benvenuto nel caso in cui l'utente sia autenticato o un link alla pagina di login nel caso in cui non lo sia.
Oltre alle cartelle già citate, seguono la cartella Tablatura, contenente la vista Tablatura.aspx, generata da TablaturaController, e la cartella Voci che contiene la vista Voci.aspx e Upload.aspx.
La vista Upload genera un form che permette di selezionare il file XML da inviare al controller VociController mentre la vista Voci si occupa di generare l'elenco delle parti che compongono la partitura.

In Voci, il ViewData.Model è dato dalla classe voci.cs, che come è stato definito precedentemente presenta la lista VociList, istanziata dal VociController.
La presenza del ViewData.Model permette di istanziare la classe voci, nel nostro caso

```
voci voci = new voci ();
```

e di assegnare a voci la proprietà ViewData.Model, nel seguente modo:

```
voci = ViewData.Model;
```

Per completare la creazione del ViewData.Model è necessario introdurre la seguente sintassi nel codice della pagina:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<Project_Tesi.Models.voci>" %>
<%@ Import Namespace="Project_Tesi.Models" %>
```

In tale maniera è possibile richiamare le componenti della classe voci, in particolare con un ciclo foreach vengono estratti i valori contenuti all'interno della lista VociList, valori che vengono mostrati all'utente.
Infine è presente la vista Tablatura.aspx che è responsabile della generazione del markup per la visualizzazione della tablatura finale.
All'interno della vista sono stati inclusi i file .js di VexFlow, che consentono di rendere graficamente la tablatura, inoltre viene istanziata la classe tablatura ed assegnatagli la proprietà ViewData.Model.
Con l'assegnazione del ViewData.Model si passa così all'estrazione, tramite cicli foreach, dei dati elaborati da TablaturaController, e sfruttando la sintassi di VexFlow si va ad ottenere finalmente la traduzione in tablatura per chitarra del codice XML in formato standard IEEE 1599.
Cliccando su Donwload PDF o Download XML vengono richiamati i relativi metodi contenuti all'interno del TablaturaController e si passa così al download del file selezionato.

CONCLUSIONI

Al termine del lavoro svolto come elaborato finale, per il corso di Laurea Triennale in Scienze e Tecnologie della Comunicazione Musicale, sono in grado di affermare che la possibilità, data dal formato XML, di poter creare una struttura ad-hoc, a seconda delle proprie esigenze, permette di poter sviluppare importanti applicazioni e di essere utile in molti e svariati campi. Sicuramente uno di questi è il campo musicale, in cui il Laboratorio d'Informatica Musicale, grazie ai professori Haus, Ludovico, Baratè e tanti altri, si è davvero affermato come perseguitore di scopi, tramutati in standard IEEE 1599, standard che propone davvero usi sconsiderati fin'ora, ma che potrebbero facilitare la didattica musicale e la presentazione di opere musicali, con l'inserimento di brani in formati audio mp3, AAC, filmati, immagini di partiture e testi, una mescolanza di elementi, che uniti tra loro formano un importante strumento di comunicazione musicale. Grazie inoltre alle tecnologie, in grande stato d'avanzamento continuo, di nuovi linguaggi di programmazione e d'interrogazione dati, l'elaborazione di file XML, la loro manipolazione e presentazione diventa molto semplice ed intuitiva ed è verso ciò che si sta orientando anche il mondo musicale.

RINGRAZIAMENTI

Dopo tre anni di studio trascorsi a Milano, con la voglia di fare e mettermi in discussione, finalmente sono giunto al traguardo della laurea triennale. Durante questi anni la mia famiglia mi è stata sempre vicina e mi ha supportato, aiutandomi nella creazione di un futuro, della mia persona. Ringrazio prima di tutto la mia famiglia, mio Padre, mia Madre, mio Fratello e mio zio Gaspare per avermi indotto verso un campo informatico-musicale, per l'affetto e la fiducia che hanno nutrito sempre in me, in qualsiasi momento della vita, nonché tutti i miei parenti, zii, nonni.

Voglio ringraziare i professori Ludovico e Baratè, per la fiducia accordatami e la disponibilità dimostrata durante questi mesi di tirocinio e sviluppo dell'elaborato finale, nonché per rappresentare una viva essenza del Laboratorio d'Informatica Musicale.

Voglio ringraziare tutti quei pazzi amici di ESN, che in questi anni si sono dimostrati una vera famiglia d'amici, in grado di ascoltarti e darti l'opportunità di esprimere ed applicare le tue idee, che mi hanno dato l'opportunità di vedere il mondo e la società da un punto di vista poliglotta e molto aperto e che sicuramente sono stati importanti per la mia crescita personale.

Infine voglio ringraziare tutti i miei amici, che purtroppo vedo poche volte all'anno, ma che rimangono pur sempre degli amici, in qualsiasi momento e istante della vita.

Grazie,

Giacomo Licari

BIBLIOGRAFIA

Luca Andrea Ludovico (2009), *IEEE 1599: a Multi-layer Approach to Music Description,* Journal of Multimedia, Vol. 4, Num. 1.

G. Haus and M. Longari (2002), *Proceedings of the First International IEEE Conference on Musical Application using XML (MAX2002),* IEEE Computer Society.

Fabio Mancini (2011), *Metodologie e Tecniche per l'Editoria Musicale,* dispensa, Milano.

Dino Esposito (2011), *Programmare ASP.NET MVC con Microsoft Visual Studio 2010*, prima Edizione, Mondadori Informatica.

Adam Freeman and Joseph Free Rattz (2010), *Pro Linq Language Integration Query in C# 2010*, Apress.

Maurizio Ma, *Codifica in MX della notazione musicale relativa alle intavolature per liuto,* LIM.

SITOGRAFIA

http://www.msdn.microsoft.com

http://www.asp.net

http://www.aspitalia.com

http://www.ludovico.it

http://www.lim.dico.unimi.it

http://www.mx.dico.unimi.it

http://www.vexflow.com

http://www.lilypond.org

APPENDICE

I CONTROLLER

VociController

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Xml.Linq;
using Project_Tesi.Models;
using System.IO;

namespace Project_Tesi.Controllers
{
   public class VociController : Controller
   {
      public ActionResult Upload()
      {
         if (!Request.IsAuthenticated)
            return RedirectToAction("LogOn", "Account");

         return View();
      }

      [HttpPost]
      public ActionResult Upload(HttpPostedFileBase file)
      {
         try
         {
            if (file != null && file.ContentLength > 0)
            {
               var fileName = Path.GetFileName(file.FileName);
               var path = Path.Combine(Server.MapPath("~/App_Data/uploads"), HttpContext.Session.SessionID.ToString() + ".xml");
               file.SaveAs(path);
               return RedirectToAction("Voci");
            }
            else
            {
               return View();
            }
         }
         catch
         {
            return View("Error");
         }
      }

      public ViewResult Voci()
      {
         List<string> obj_misure = null;
         voci voci;

          // Caricamento Documento
         string filename = HttpContext.Session.SessionID.ToString() + ".xml"; //sarebbe il file caricato dall'utente


         XDocument doc_intero = XDocument.Load(Request.MapPath("~/App_Data/uploads/" + filename));


         IEnumerable<XElement> voices = from v in doc_intero.Root.Descendants("part")
                                        select v;
```

```
            obj_misure = new List<string>();
            voci = new voci();
            voci.sessione = HttpContext.Session.SessionID.ToString();

            foreach(var v in voices)
            {
               obj_misure.Add(v.Attribute("id").Value);
               voci.VociList = obj_misure;
            }

            return View(voci);
        }

    }
}
```

TablaturaController

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Xml.Linq;
using Project_Tesi.Models;
using System.IO;
using System.Configuration;
using System.Diagnostics;

namespace Project_Tesi.Controllers
{
   public class TablaturaController : Controller
   {
      public ViewResult Tablatura(string id, string sessione)
      {
         tablatura tablatura;

         List<misura> obj_misure = null;
         List<accordo> obj_accordi;
         List<nota> obj_note;

         misura obj_misura;
         accordo obj_accordo;
         nota obj_nota;

         posizione pos;

         string saccidental;
         string id_part = id;
         string sessione_tab = sessione;
         sessione_tab = sessione_tab + ".xml";


         XDocument doc_intero = XDocument.Load(Request.MapPath("~/App_Data/uploads/" + sessione_tab));

         ////////////////////////////
         // Controllo Parte //
         ////////////////////////
         IEnumerable<XElement> control_part = from c in doc_intero.Root.Descendants("part")
                                              where (string)c.Attribute("id") == id_part
```

```
                                select c;

        foreach (XElement x in control_part)
        {
            IEnumerable<XElement> control_measure = from c in control_part.Descendants("measure")
                                    select c;

            foreach (XElement misura_c in control_measure)
            {
                IEnumerable<XElement> control_chord = from c in misura_c.Descendants("chord")
                                    select c;

                foreach (XElement chord_c in control_chord)
                {
                    IEnumerable<XElement> control_pitch = from c in chord_c.Descendants("pitch")
                                        select c;
                    foreach (XElement pitch_c in control_pitch)
                    {
                        switch ((int)pitch_c.Attribute("octave"))
                        {
                            case 1:
                                ViewData["Controllo"] = "La partitura potrebbe non essere per chitarra, pertanto potrebbero visualizzarsi degli
errori di traduzione.";
                                break;
                            case 2:
                                ViewData["Controllo"] = "La partitura potrebbe non essere per chitarra, pertanto potrebbero visualizzarsi degli
errori di traduzione.";
                                break;
                            case 8:
                                ViewData["Controllo"] = "La partitura potrebbe non essere per chitarra, pertanto potrebbero visualizzarsi degli
errori di traduzione.";
                                break;
                            default:
                                break;
                        }
                    }
                }
            }
        }


        ////////////////////////////////////
        // RECUPERO NOTE ///
        ////////////////////////////////////
        chitarra ichitarra = new chitarra(); //istanzio la classe chitarra.cs

        IEnumerable<XElement> voices = from v in doc_intero.Root.Descendants("part")
                                where (string)v.Attribute("id") == id_part
                                select v;
        var main_title = from t in doc_intero.Root.Descendants("main_title")
                         select t;
        var author = from a in doc_intero.Root.Descendants("author")
                     where (string)a.Attribute("type") == "composer"
                     select a;

        tablatura = new tablatura();
        foreach (var t in main_title)
        {
            tablatura.titolo = t.Value.ToString();
        }
        foreach (var a in author)
        {
            tablatura.autore = a.Value.ToString();
        }

        foreach (XElement voice in voices)
        {
```

```
            IEnumerable<XElement> measure = from m in voice.Descendants("measure")
                                            select m; //seleziona tutte le misure

            obj_misure = new List<misura>();

            foreach (XElement misura in measure)
            {
               obj_misura = new misura();
               IEnumerable<XElement> chords = from p in misura.Descendants("chord")
                                              select p; //seleziona tutti i chord

               obj_accordi = new List<accordo>();

               foreach (XElement chord in chords)
               {
                  obj_accordo = new accordo();

                  var durate = from a in chord.Descendants("duration")
                               select a.Attribute("den");

                  var event_ref = from a in misura.Descendants("chord")
                                  select a;

                  foreach (var durata in durate)
                  {
                     obj_accordo.durata = Int32.Parse(durata.Value);
                  }

                  foreach(var x in event_ref)
                  {
                     obj_accordo.event_ref = x.Attribute("event_ref").Value;
                  }


                  var note = from p in chord.Descendants("pitch")
                             select new { step = p.Attribute("step"), octave = p.Attribute("octave"), accidental =
(string)p.Attribute("actual_accidental") ?? "natural" }; //seleziono step, ottava e alterazione


                  obj_note = new List<nota>();

                  foreach (var nota in note)
                  {
                     obj_nota = new nota();
                     obj_nota.ottava = Int32.Parse(nota.octave.Value);
                     obj_nota.step = nota.step.Value;

                     saccidental = nota.accidental.ToString();
                     switch (saccidental)
                     {
                        case "flat":
                           obj_nota.accidental = accidental.flat;
                           break;

                        case "sharp":
                           obj_nota.accidental = accidental.sharp;
                           break;

                        case "natural":
                           obj_nota.accidental = accidental.natural;
                           break;
                     }


                     pos = ichitarra.get_posizione(obj_note, obj_nota.ottava, obj_nota.step, (int)(obj_nota.accidental));
                     obj_nota.corda = pos.corda;
```

```
            obj_nota.tasto = pos.tasto;

            obj_note.Add(obj_nota);

        }

        obj_accordo.note = obj_note;

        obj_accordi.Add(obj_accordo);
      }
      obj_misura.accordi = obj_accordi;
      obj_misure.Add(obj_misura);
    }
    tablatura.misure = obj_misure;
}

////////////////////////////////////////
//Creazione XML tablatura//
////////////////////////////////////////
string outfile = Server.MapPath("/App_Data/uploads/TAB" + sessione + ".xml");

IEnumerable<XElement> creazione_staff = from d in doc_intero.Root.Descendants("staff_list")
                                        where (string)d.Element("staff").Attribute("id") == id_part + "_staff"
                                        select d.Element("staff");

foreach (XElement staff_tab in creazione_staff)
{
    ////////////////////////////////////////////////////////////////////////
    // Creo i tag da inserire all'interno dello staff_list ///
    ////////////////////////////////////////////////////////////////////////
    XElement tablature_tuning = new XElement("tablature_tuning");
    XAttribute type = new XAttribute("type", "E");

    XElement strings1 = new XElement("string");
    XAttribute string_number1 = new XAttribute("string_number", 1);
    XAttribute string_pitch1 = new XAttribute("string_pitch", "E");
    XAttribute octave1 = new XAttribute("string_octave", 3);

    XElement strings2 = new XElement("string");
    XAttribute string_number2 = new XAttribute("string_number", 2);
    XAttribute string_pitch2 = new XAttribute("string_pitch", "A");
    XAttribute octave2 = new XAttribute("string_octave", 3);

    XElement strings3 = new XElement("string");
    XAttribute string_number3 = new XAttribute("string_number", 3);
    XAttribute string_pitch3 = new XAttribute("string_pitch", "D");
    XAttribute octave3 = new XAttribute("string_octave", 4);

    XElement strings4 = new XElement("string");
    XAttribute string_number4 = new XAttribute("string_number", 4);
    XAttribute string_pitch4 = new XAttribute("string_pitch", "G");
    XAttribute octave4 = new XAttribute("string_octave", 4);

    XElement strings5 = new XElement("string");
    XAttribute string_number5 = new XAttribute("string_number", 5);
    XAttribute string_pitch5 = new XAttribute("string_pitch", "B");
    XAttribute octave5 = new XAttribute("string_octave", 4);

    XElement strings6 = new XElement("string");
    XAttribute string_number6 = new XAttribute("string_number", 6);
    XAttribute string_pitch6 = new XAttribute("string_pitch", "E");
    XAttribute octave6 = new XAttribute("string_octave", 5);

    tablature_tuning.Add(type);
    strings1.Add(string_number1, string_pitch1, octave1);
    strings2.Add(string_number2, string_pitch2, octave2);
    strings3.Add(string_number3, string_pitch3, octave3);
```

```
    strings4.Add(string_number4, string_pitch4, octave4);
    strings5.Add(string_number5, string_pitch5, octave5);
    strings6.Add(string_number6, string_pitch6, octave6);
    tablature_tuning.Add(strings1);
    tablature_tuning.Add(strings2);
    tablature_tuning.Add(strings3);
    tablature_tuning.Add(strings4);
    tablature_tuning.Add(strings5);
    tablature_tuning.Add(strings6);
    staff_tab.Add(tablature_tuning);
}

////////////////////////////////////////////////////////////////////////////
//Creo i tag per l'inserimento dell'informazione musicale //
////////////////////////////////////////////////////////////////////////////
int count_measure = 1;
foreach (misura m in obj_misure)
{
    IEnumerable<XElement> voice_tab = from d in doc_intero.Root.Descendants("measure")
                                      where (string)d.Attribute("number") == count_measure.ToString()
                                      select d.Element("voice");
    foreach (XElement x in voice_tab)
    {
        XElement tab_finale = new XElement("default");

        foreach (accordo a in m.accordi)
        {
            XElement tablature_symbol = new XElement("tablature_symbol");
            XAttribute event_ref = new XAttribute("event_ref", a.event_ref);


            XAttribute num = new XAttribute("num", 1);
            XAttribute den = new XAttribute("den", a.durata);

            XElement duration = new XElement("duration");
            duration.Add(num, den);
            tablature_symbol.Add(duration);

            foreach (nota n in a.note)
            {
                XElement key = new XElement("key");
                XElement tablature_pitch = new XElement("tablature_pitch");
                XAttribute string_number = new XAttribute("string_number", n.corda);
                XAttribute key_number = new XAttribute("key_number", n.tasto);

                tablature_pitch.Add(string_number, key_number);
                key.Add(tablature_pitch);
                tablature_symbol.Add(key);
                tab_finale = tablature_symbol;
            }
            tablature_symbol.Add(event_ref);
            x.Add(tablature_symbol);
        }

    }
    count_measure++;
}

doc_intero.Save(outfile);

////////////////////////////////////////////////
// Creazione PDF con Lilypond //
////////////////////////////////////////////////
string lilypond = ConfigurationManager.AppSettings["lilypond"];                 //si riferisce al Key di Web.Config
string ifile = Server.MapPath("/App_Data/uploads/" + sessione + ".ly");          //file di input
string ofilename = Server.MapPath("/App_Data/uploads/" + sessione);              //nome del file di output senza estensione
string ofile = Server.MapPath("/App_Data/uploads/" + sessione + ".pdf");         //file di output
```

```
    string lilypondContent = "";

    try
    {
        StreamWriter LilypondFile = new StreamWriter(ifile);
        lilypondContent = "\\version \"2.10.0\"" + Environment.NewLine;
        lilypondContent += "\\header {" + Environment.NewLine;
        lilypondContent += "tagline = ##f" + Environment.NewLine;
        lilypondContent += "title = " + "\"" + tablatura.titolo + "\"" + Environment.NewLine;
        lilypondContent += "arranger = " + "\"" + tablatura.autore + "\"" + Environment.NewLine;
        lilypondContent += "}" + Environment.NewLine;
        lilypondContent += "music ={" + Environment.NewLine; //inizia la parte di scrittura della tablatura
        lilypondContent += "\\time 4/4" + Environment.NewLine;

        // scrittura accordi
        int misure_riga = 4;
        int count = 1;
        foreach (misura m in obj_misure)
        {
            foreach (accordo a in m.accordi)
            {
                lilypondContent += get_lilypond_chord(a) + Environment.NewLine;
            }
            if (count % misure_riga == 0)
            {
                if (obj_misure.Count() == count)
                    lilypondContent += "\\bar \"|.\"" + Environment.NewLine;
                else
                    lilypondContent += "\\break" + Environment.NewLine;
                //count = 0;
            }
            count++;
        }
        if (obj_misure.Count() == count - 1)
            lilypondContent += "\\bar \"|.\"" + Environment.NewLine;

        lilypondContent += "}" + Environment.NewLine;
        lilypondContent += "\\new TabStaff {" + Environment.NewLine;
        lilypondContent += "\\music" + Environment.NewLine;
        lilypondContent += "}";

        LilypondFile.Write(lilypondContent);
        LilypondFile.Close();

        ProcessStartInfo startLilypond = new ProcessStartInfo();
        startLilypond.CreateNoWindow = false;
        startLilypond.UseShellExecute = true;
        startLilypond.FileName = lilypond; //indica l'eseguibile
        startLilypond.WindowStyle = ProcessWindowStyle.Hidden;
        startLilypond.Arguments = @"-o """ + ofilename + @""" """ + ifile + @"""";

        using (Process exeProcess = Process.Start(startLilypond))
        {
            exeProcess.WaitForExit();
        }
    }
    catch (Exception e)
    {
        ViewData["Title"] = "Siamo spiacenti, si è verificato un errore.";
        ViewData["Description"] = e.Message;

        return View("Error");
    }

    return View(tablatura);
}
//< e,\6 a,\5 d\4 g\3 b\2 e'\1 > 4 promemoria sintassi lilypond
```

```
        private string get_lilypond_chord(accordo a)
        {
            string accordo = "";
            accordo = "< ";
            foreach (nota n in a.note)
            {
                accordo += n.step.ToLower() + get_lilypond_alterazione(n.accidental) + get_lilypond_ottava(n.ottava) + "\\" +
n.corda.ToString() + " ";
            }
            accordo += "> " + a.durata.ToString();
            return accordo;
        }

        private string get_lilypond_alterazione(accidental a)
        {
            string alterazione = "";
            switch (a)
            {
                case accidental.flat:
                    alterazione = "es";
                    break;
                case accidental.sharp:
                    alterazione = "is";
                    break;

                default:
                    alterazione = "";
                    break;
            }

            return alterazione;

        }
        private string get_lilypond_ottava(int o)
        {
            string ottava = "";
            switch (o)
            {
                case 3:
                    ottava = ",";
                    break;
                case 4:
                    ottava = "";
                    break;
                case 5:
                    ottava = "'";
                    break;
                case 6:
                    ottava = "''";
                    break;
                case 7:
                    ottava = "'''";
                    break;
                case 8:
                    ottava = "''''";
                    break;

            }
            return ottava;
        }

        public ActionResult Download_PDF()
        {
            string pdf = "~/App_Data/uploads/" + Session.SessionID + ".pdf";

            Response.Clear();
            Response.Buffer = true;
```

```
            Response.ContentType = "application/pdf";
            Response.AddHeader("Content-Disposition", "attachment; filename=\"" + Path.GetFileName(pdf) + "\"");
            Response.TransmitFile(pdf);

            return View();

        }

        public ActionResult Download_XML()
        {
            string XML_file = "~/App_Data/uploads/TAB" + Session.SessionID + ".xml";

            Response.Clear();
            Response.Buffer = true;

            Response.ContentType = "application/html+xml";
            Response.AddHeader("Content-Disposition", "attachment; filename=\"" + Path.GetFileName(XML_file) + "\"");
            Response.TransmitFile(XML_file);

            return View();
        }

    }
}
```

I MODELS

Global

```
public enum accidental { flat = -1, natural = 0, sharp = 1 }

public struct posizione
{
   public int corda;
   public int tasto;
}

public struct pitch
{
   public int ottava;
   public string step;
   public string accidental;

}
```

Chitarra

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Xml.Linq;
using System.IO;

namespace Project_Tesi.Models
{
   public class chitarra
   {

     public XDocument data;

      public chitarra()
      {
         var fileXML = AppDomain.CurrentDomain.BaseDirectory + "/App_Data/chitarra.xml";

         data = XDocument.Load(fileXML);

      }
      public posizione get_posizione(List<nota> note, int ottava, string nota, int alterazione) //metodo
      {
         Boolean trovato = false;
         int LARGHEZZA_ACCORDO = 4; //fa sì che venga scelto un accordo entro n tasti

         IEnumerable<XElement> iottava = from s in data.Descendants("ottava")
                                         where (string)s.Attribute("id") == ottava.ToString()
                                         select s;


         IEnumerable<XElement> inota = from s in iottava.Descendants("nota")
                                       where (string)s.Attribute("id") == nota
                                       select s;


         IEnumerable<XElement> ialterazione = from s in inota.Descendants("alterazione")
                                              where (string)s.Attribute("id") == alterazione.ToString()
```

```
                                select s;

        IEnumerable<XElement> iposizione = from s in ialterazione.Descendants("posizione")
                                select s;

        posizione p;
        p.corda = 0;
        p.tasto = 0;

        foreach (XElement x in iposizione)
        {

            p.corda = Int32.Parse( x.Element("corda").Value);
            p.tasto = Int32.Parse( x.Element("tasto").Value);

            /* scelta posizione migliore */
            foreach (nota n in note)
            {
                if (n.corda != p.corda)
                {
                    if (Math.Abs(p.tasto - n.tasto) <= LARGHEZZA_ACCORDO)  //Math.Abs() calcola valore assoluto della differenza
                    {
                        trovato = true;
                        break;
                    }
                }
            }

            if (trovato)
                break;
        }
        return p;
    }

}
}
```

LE VIEWS

Voci

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<Project_Tesi.Models.voci>" %>
<%@ Import Namespace="Project_Tesi.Models" %>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
            Voci
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
<link href="../../Content/Voci.css" rel="stylesheet" type="text/css" />

<div id="home">
<div id="voci">
    <h2>Clicca sulla voce desiderata per ottenerne la tablatura.</h2>
    <p>
        <!--<h2>Utente <%: Page.User.Identity.Name %></h2>-->
        <table id="gridVoci">
            <tr>
                <th scope="col">Voce</th>
            </tr>
            <%
                voci voci = new voci();
                voci = ViewData.Model;
                foreach (string x in voci.VociList)
                { %>
                <tr><td><%: Html.ActionLink(x.ToString(),"Tablatura","Tablatura",new { id = x, sessione = voci.sessione}, null)
%></td></tr>
            <% } %>
        </table>
    </p>
    </div>
    <div id="logo_voci">
        <img src="../../Content/images/unimi.jpg" />
    </div>
    </div>
</asp:Content>
```

Tablatura

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<Project_Tesi.Models.tablatura>" %>
<%@ Import Namespace="Project_Tesi.Models" %>
<asp:Content ID="tablaturaTitle" ContentPlaceHolderID="TitleContent" runat="server">
    Tablatura
</asp:Content>

<asp:Content ID="TablaturaContent" ContentPlaceHolderID="MainContent" runat="server">
    <link href="../../Content/Tablatura.css" rel="stylesheet" type="text/css" />
    <link href="../../vexflow/tabdiv/tabdiv.css" media="screen" rel="Stylesheet" type="text/css" />
    <!--<link href="../../vexflow/tabdiv/style.css" media="screen" rel="Stylesheet" type="text/css" /> -->

    <!-- VexFlow Uncompiled Sources -->

    <!-- Support Sources -->
    <script src="../../vexflow/vextab/support/jquery.js"></script>
    <script src="../../vexflow/vextab/support/raphael.js"></script>

    <!-- TabDiv Sources -->
    <script src="../../vexflow/tabdiv/tabdiv.js"></script>

    <!-- VexFlow Compiled Sources -->
    <script src="../../vexflow/vexflow/vexflow.js"></script>
    <script src="../../vexflow/vexflow/vextabdiv.js"></script>


<%
    tablatura tablatura = new tablatura();
    string s = "";

    int NUM_MISURE_RIGA = 4;
    int i = 0;

    tablatura = ViewData.Model;
    foreach (misura m in tablatura.misure)
    {
        /* se la misura è vuota non scrivo */
        if (m.accordi.Count > 0)
        {
            /* inserisco nuova riga ogni NUM_MISURE_RIGHE */
            if (i % NUM_MISURE_RIGA == 0)
            {
                s = s + "tabstave" + Environment.NewLine.ToString();
            }

            /* compongo le note da scrivere */
            foreach (accordo a in m.accordi)
            {
                s = s + "notes (";
                foreach (nota n in a.note)
                {
                    s = s + n.tasto + "/" + n.corda + ".";
                }
                s = s.Substring(0, s.Length - 1) + ")" + Environment.NewLine.ToString();
            }

            i++;

            /* se ultima misura della riga non scrivo la barretta */
            if (i % NUM_MISURE_RIGA == 0)
                s = s.Substring(0, s.Length - 2) + Environment.NewLine.ToString();
            else
                s = s.Substring(0, s.Length - 2) + " |" + Environment.NewLine.ToString();
        }
```

```
    }

%>
    <!--<h2>Utente <%: Page.User.Identity.Name %></h2>-->
    <div id="download">
        <img src="../../Content/images/pdf.png" style="width:23%" />
        <%: Html.ActionLink("Download PDF", "Download_PDF", "Tablatura")%>
    </div>
    <div id="download_xml">
        <img src="../../Content/images/xmllogo.png" style="width:30%" />
        <%: Html.ActionLink("Download XML", "Download_XML","Tablatura") %>
    </div>

    <div id="tablatura">
    <h2>Titolo <i>"<%: tablatura.titolo %>
                                <% if (tablatura.titolo == "")
                                 {
                                    Response.Write("non definito");

                                 } %>"</i>

    </h2>
    <h3>Autore <i>
            <u>
             <%: tablatura.autore %>
             <% if (tablatura.autore == "")
                {
                    Response.Write("sconosciuto");
                } %>
            </u>

    </i></h3>
    </div>
    <div class="header">
    <p id="controllo">
        <%: ViewData["Controllo"] %>
    </p>
        <div id="tablatura" class="vex-tabdiv" width="880" scale="0.9">
        <%: s%>
        </div>
    </div>
</asp:Content>
```